ML_Project

December 22, 2022

[195]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import scipy
import seaborn as sns
```

[ ]:
```

```

Here we have downloaded a dataset from this link in

kaggle:https://www.kaggle.com/code/yasirakyzl/covid-19-ml-model-90-accuracy/data . Our goal as part of this project is to design an algorithm to predict the risk of a patient dying from Covid based on their medical data and history.

[196]:
```
df = pd.read_csv("covidData.csv");
```

In the Dataset that we are using the column date died contains the dates at which the patient had died. If the patient is still alive then the Date_Died column contains the value '9999-99-99'. So in this step we are converting the date_died column into a binary column with 1(dead) and 0(alive) as the possible values.

[197]:
```
# df = df.replace(to_replace=97, value=np.nan).dropna()
# df = df.replace(to_replace=99, value=np.nan).dropna()

df['DEATH'] = np.where(df['DATE_DIED'] == '9999-99-99', 0, 1)
df = df.drop('DATE_DIED', axis=1)
```

Next we split the dataset into training and test data.

[198]:
```
from sklearn.model_selection import train_test_split

X = df.drop(columns="DEATH")
y = df["DEATH"]

Xtr, Xts, ytr, yts = train_test_split(X,y,test_size=0.2,random_state=42)
print("Train x :",Xtr.shape)
print("Test x :",Xts.shape)
print("Train y :",ytr.shape)
print("Test y :",yts.shape)
```

```
X
```

```
Train x : (838860, 20)
Test x : (209715, 20)
Train y : (838860,)
Test y : (209715,)
```

[198]:

```
         USMER  MEDICAL_UNIT  SEX  PATIENT_TYPE  INTUBED  PNEUMONIA  AGE  \
0            2             1    1             1       97          1   65
1            2             1    2             1       97          1   72
2            2             1    2             2        1          2   55
3            2             1    1             1       97          2   53
4            2             1    2             1       97          2   68
…          …             …    …             …        …          …    …
1048570      2            13    2             1       97          2   40
1048571      1            13    2             2        2          2   51
1048572      2            13    2             1       97          2   55
1048573      2            13    2             1       97          2   28
1048574      2            13    2             1       97          2   52

         PREGNANT  DIABETES  COPD  ASTHMA  INMSUPR  HIPERTENSION  \
0               2         2     2       2        2             1
1              97         2     2       2        2             1
2              97         1     2       2        2             2
3               2         2     2       2        2             2
4              97         1     2       2        2             1
…             …         …    …      …        …             …
1048570        97         2     2       2        2             2
1048571        97         2     2       2        2             1
1048572        97         2     2       2        2             2
1048573        97         2     2       2        2             2
1048574        97         2     2       2        2             2

         OTHER_DISEASE  CARDIOVASCULAR  OBESITY  RENAL_CHRONIC  TOBACCO  \
0                    2               2        2              2        2
1                    2               2        1              1        2
2                    2               2        2              2        2
3                    2               2        2              2        2
4                    2               2        2              2        2
…                  …               …        …              …        …
1048570              2               2        2              2        2
1048571              2               2        2              2        2
1048572              2               2        2              2        2
1048573              2               2        2              2        2
1048574              2               2        2              2        2
```

```
         CLASIFFICATION_FINAL  ICU
0                           3   97
1                           5   97
2                           3    2
3                           7   97
4                           3   97
…                       …   …
1048570                     7   97
1048571                     7    2
1048572                     7   97
1048573                     7   97
1048574                     7   97

[1048575 rows x 20 columns]
```

Now we create a linear regresion model on the dataset and check its accuracy

[199]:
```
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(Xtr,ytr)
print("Linear Regression Accuracy :",linreg.score(Xts, yts))
```

```
Linear Regression Accuracy : 0.30115335137646615
```

You can see that the accuracy obtained for linear regression is very low. This is because our problem is a classification problem and not a regression problem. So we instead we use logistic regression to create our model.

[201]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(Xtr,ytr)
print("Logistic Regression Accuracy :",logreg.score(Xts, yts))
```

```
Logistic Regression Accuracy : 0.9356889111413108
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

We now how our prediction algorithm performs if we use a Decison Tree instead of logistic Regression

[202]:
```python
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(Xtr,ytr)
print("Decision Tree :",dt.score(Xts, yts))
```

```
Decision Tree : 0.9398278616217247
```

We notice that the model using the decision tree performs better than logistic regression this because our dataset is large . Furthermore logistic regression works best only for simple datasets which are small and linearly seperable. For larger datasets more complex models are generally preferred. We will next see how welldoes the random forest model which makes use of multiple decision trees work on this dataset.

[203]:
```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(Xtr,ytr)
print("Random forrest Accuracy :",rf.score(Xts, yts))
```

```
Random forrest Accuracy : 0.9447822044202847
```

We will see how well a prediction algorithm using a nueral network works on the test data. For the neural network which we have built. We are using an input layer a hiden layer and an output layer. Since there are 20 input features in our dataset we are using (2*20 -1) hidden units(neurons) in each hidden layer. 1 output unit the number of epochs is 30. The activation function chosen is relu for the hidden layers and sigmoid for the output layer. We chose these hyperparameters after reading this article https://medium.com/codex/how-to-tune-hyperparameters-for-better-neural-network-performance-b8f542855d2e on medium.

[204]:
```python
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation

nin = Xtr.shape[1]
nh = 39
nout = 1
model = Sequential()
model.add(Dense(units=nh, input_shape=(nin,), activation='relu', name='hidden'))
model.add(Dense(units=nh, activation='relu', name='hidden2'))
model.add(Dense(units=nout, activation='sigmoid', name='output'))
```

[207]:
```python
from tensorflow.keras import optimizers

opt = optimizers.Adam(lr=0.001)
model.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.8/dist-
packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
```

```
super(Adam, self).__init__(name, **kwargs)
```

[208]:
```
hist = model.fit(Xtr, ytr, epochs=30, batch_size=100, validation_data=(Xts,yts))
```

```
Epoch 1/30
8389/8389 [==============================] - 22s 2ms/step - loss: 0.1434 -
accuracy: 0.9379 - val_loss: 0.1290 - val_accuracy: 0.9452
Epoch 2/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1260 -
accuracy: 0.9457 - val_loss: 0.1203 - val_accuracy: 0.9483
Epoch 3/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1217 -
accuracy: 0.9473 - val_loss: 0.1192 - val_accuracy: 0.9478
Epoch 4/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1200 -
accuracy: 0.9476 - val_loss: 0.1196 - val_accuracy: 0.9495
Epoch 5/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1189 -
accuracy: 0.9482 - val_loss: 0.1195 - val_accuracy: 0.9481
Epoch 6/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1183 -
accuracy: 0.9484 - val_loss: 0.1198 - val_accuracy: 0.9485
Epoch 7/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1179 -
accuracy: 0.9486 - val_loss: 0.1181 - val_accuracy: 0.9491
Epoch 8/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1177 -
accuracy: 0.9486 - val_loss: 0.1158 - val_accuracy: 0.9501
Epoch 9/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1174 -
accuracy: 0.9487 - val_loss: 0.1149 - val_accuracy: 0.9502
Epoch 10/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1171 -
accuracy: 0.9488 - val_loss: 0.1196 - val_accuracy: 0.9494
Epoch 11/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1170 -
accuracy: 0.9488 - val_loss: 0.1206 - val_accuracy: 0.9480
Epoch 12/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1168 -
accuracy: 0.9489 - val_loss: 0.1157 - val_accuracy: 0.9498
Epoch 13/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1166 -
accuracy: 0.9490 - val_loss: 0.1156 - val_accuracy: 0.9499
Epoch 14/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1165 -
accuracy: 0.9491 - val_loss: 0.1154 - val_accuracy: 0.9501
Epoch 15/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1164 -
```

```
accuracy: 0.9493 - val_loss: 0.1162 - val_accuracy: 0.9502
Epoch 16/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1162 -
accuracy: 0.9492 - val_loss: 0.1152 - val_accuracy: 0.9504
Epoch 17/30
8389/8389 [==============================] - 23s 3ms/step - loss: 0.1161 -
accuracy: 0.9493 - val_loss: 0.1147 - val_accuracy: 0.9499
Epoch 18/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1160 -
accuracy: 0.9492 - val_loss: 0.1150 - val_accuracy: 0.9502
Epoch 19/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1158 -
accuracy: 0.9493 - val_loss: 0.1160 - val_accuracy: 0.9499
Epoch 20/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1157 -
accuracy: 0.9493 - val_loss: 0.1146 - val_accuracy: 0.9501
Epoch 21/30
8389/8389 [==============================] - 20s 2ms/step - loss: 0.1157 -
accuracy: 0.9495 - val_loss: 0.1144 - val_accuracy: 0.9502
Epoch 22/30
8389/8389 [==============================] - 21s 3ms/step - loss: 0.1156 -
accuracy: 0.9494 - val_loss: 0.1175 - val_accuracy: 0.9491
Epoch 23/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1155 -
accuracy: 0.9495 - val_loss: 0.1151 - val_accuracy: 0.9499
Epoch 24/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1154 -
accuracy: 0.9495 - val_loss: 0.1155 - val_accuracy: 0.9494
Epoch 25/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1153 -
accuracy: 0.9496 - val_loss: 0.1144 - val_accuracy: 0.9503
Epoch 26/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1152 -
accuracy: 0.9496 - val_loss: 0.1167 - val_accuracy: 0.9498
Epoch 27/30
8389/8389 [==============================] - 22s 3ms/step - loss: 0.1152 -
accuracy: 0.9497 - val_loss: 0.1154 - val_accuracy: 0.9502
Epoch 28/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1152 -
accuracy: 0.9496 - val_loss: 0.1143 - val_accuracy: 0.9503
Epoch 29/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1150 -
accuracy: 0.9498 - val_loss: 0.1138 - val_accuracy: 0.9509
Epoch 30/30
8389/8389 [==============================] - 21s 2ms/step - loss: 0.1149 -
accuracy: 0.9498 - val_loss: 0.1145 - val_accuracy: 0.9500
```

[209]:
```
score, acc = model.evaluate(Xts, yts)
print('Test accuracy:', acc)
```

```
6554/6554 [==============================] - 10s 2ms/step - loss: 0.1145 -
accuracy: 0.9500
Test accuracy: 0.9500464797019958
```

We have run all our models . However we can still improve the accuracy by doing some feature selection to remove some irrelevant features which might be reducing the accuracy of our model. Furthermore our dataset contains some values which are actually null values which migth be affecting our overall result. In the dataset we used the null values are 97 and 99.

[210]:
```
q1=df.columns
q1
```

[210]:
```
Index(['USMER', 'MEDICAL_UNIT', 'SEX', 'PATIENT_TYPE', 'INTUBED', 'PNEUMONIA',
       'AGE', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA', 'INMSUPR',
       'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR', 'OBESITY',
       'RENAL_CHRONIC', 'TOBACCO', 'CLASIFFICATION_FINAL', 'ICU', 'DEATH'],
      dtype='object')
```

Before any feature selection is done we see the count of the unique values for each feature in the dataset.

[211]:
```
df['USMER'].value_counts()
```

[211]:
```
2    662903
1    385672
Name: USMER, dtype: int64
```

[212]:
```
df['MEDICAL_UNIT'].value_counts()
```

[212]:
```
12    602995
4     314405
6      40584
9      38116
3      19175
8      10399
10      7873
5       7244
11      5577
13       996
7        891
2        169
1        151
Name: MEDICAL_UNIT, dtype: int64
```

[213]:
```
df['SEX'].value_counts()
```

[213]:
```
1    525064
2    523511
Name: SEX, dtype: int64
```

[214]:
```
df['PATIENT_TYPE'].value_counts()
```

[214]:
```
1    848544
2    200031
Name: PATIENT_TYPE, dtype: int64
```

[215]:
```
df['INTUBED'].value_counts()
```

[215]:
```
97    848544
2     159050
1      33656
99      7325
Name: INTUBED, dtype: int64
```

[216]:
```
df['PNEUMONIA'].value_counts()
```

[216]:
```
2     892534
1     140038
99     16003
Name: PNEUMONIA, dtype: int64
```

[217]:
```
df['AGE'].value_counts()
```

[217]:
```
30     27010
31     25927
28     25313
29     25134
34     24872
       …
114        2
116        2
111        1
121        1
113        1
Name: AGE, Length: 121, dtype: int64
```

[218]:
```
df['PREGNANT'].value_counts()
```

[218]:
```
97    523511
2     513179
1       8131
98      3754
Name: PREGNANT, dtype: int64
```

[219]:
```
df['DIABETES'].value_counts()
```

[219]:
```
2     920248
1     124989
98      3338
Name: DIABETES, dtype: int64
```

[220]:
```
df['COPD'].value_counts()
```

[220]:
```
2     1030510
1       15062
98       3003
Name: COPD, dtype: int64
```

[221]:
```
df['ASTHMA'].value_counts()
```

[221]:
```
2     1014024
1       31572
98       2979
Name: ASTHMA, dtype: int64
```

[222]:
```
df['INMSUPR'].value_counts()
```

[222]:
```
2     1031001
1       14170
98       3404
Name: INMSUPR, dtype: int64
```

[223]:
```
df['HIPERTENSION'].value_counts()
```

[223]:
```
2     882742
1     162729
98      3104
Name: HIPERTENSION, dtype: int64
```

[224]:
```
df['OTHER_DISEASE'].value_counts()
```

[224]:
```
2     1015490
1       28040
98       5045
Name: OTHER_DISEASE, dtype: int64
```

[225]:
```
df['CARDIOVASCULAR'].value_counts()
```

[225]:
```
2     1024730
1       20769
98       3076
```

```
Name: CARDIOVASCULAR, dtype: int64
```

[226]:
```
df['OBESITY'].value_counts()
```

[226]:
```
2     885727
1     159816
98      3032
Name: OBESITY, dtype: int64
```

[227]:
```
df['RENAL_CHRONIC'].value_counts()
```

[227]:
```
2     1026665
1       18904
98       3006
Name: RENAL_CHRONIC, dtype: int64
```

[228]:
```
df['TOBACCO'].value_counts()
```

[228]:
```
2     960979
1      84376
98      3220
Name: TOBACCO, dtype: int64
```

[229]:
```
df['CLASIFFICATION_FINAL'].value_counts()
```

[229]:
```
7    499250
3    381527
6    128133
5     26091
1      8601
4      3122
2      1851
Name: CLASIFFICATION_FINAL, dtype: int64
```

[230]:
```
df['ICU'].value_counts()
```

[230]:
```
97    848544
2     175685
1      16858
99      7488
Name: ICU, dtype: int64
```

We notice that the features INTUBED and ICU have more than 50% of the data as null(value is 97 or 99). So we cant simply remove all rows having atleast one feature having 97 or 99 in the dataset as we wil end up losing a lot of data. However since both INTUBED and ICU are binary attributes we can use logistic regression to fill the null value in INTUBED and ICU. After this done we can delete all the rows having null values in atleast one feature. For dealing with situation above we made use of a similar technique found in https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a-complete-guide/.

[231]:
```
from numpy.ma.core import filled
df_mod = df
df_mod= df.loc[(df['ICU'] != 97) & (df['ICU'] != 99)]

df_to_fill = df.loc[(df['ICU'] == 99) | (df['ICU'] == 97)]


from sklearn.linear_model import LogisticRegression
X_mod = df_mod.loc[:, df.columns != 'ICU']
y_mod = df_mod['ICU']

logreg = LogisticRegression()
logreg.fit(X_mod, y_mod)




df_to_fill['ICU']  = logreg.predict(df_to_fill.loc[:, df.columns != 'ICU'])


filled_df = df_to_fill.append(df_mod)

df = filled_df
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
<ipython-input-231-3cf56cb4e357>:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_to_fill['ICU']  = logreg.predict(df_to_fill.loc[:, df.columns != 'ICU'])
```

[232]:
```
from numpy.ma.core import filled
df_mod_int = df.loc[(df['INTUBED'] != 97) & (df['INTUBED'] != 99)]
```

```
df_to_fill_int = df.loc[(df['INTUBED'] == 99) | (df['INTUBED'] == 97)]


from sklearn.linear_model import LogisticRegression
X_mod = df_mod_int.loc[:, df.columns != 'INTUBED']
y_mod = df_mod_int['INTUBED']

logreg = LogisticRegression()
logreg.fit(X_mod, y_mod)





df_to_fill_int['INTUBED']  = logreg.predict(df_to_fill_int.loc[:, df.columns !=␣
↪'INTUBED'])


filled_df_int = df_to_fill_int.append(df_mod_int)

df =filled_df_int
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
<ipython-input-232-838cdcc7955c>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_to_fill_int['INTUBED']  = logreg.predict(df_to_fill_int.loc[:, df.columns
!= 'INTUBED'])
```

[233]:
```
df = df.replace(to_replace=97, value=np.nan).dropna()
df = df.replace(to_replace=99, value=np.nan).dropna()
X = df.drop(columns="DEATH")
y = df["DEATH"]

Xtr, Xts, ytr, yts = train_test_split(X,y,test_size=0.2,random_state=42)
print("Train x :",Xtr.shape)
```

```
print("Test x :",Xts.shape)
print("Train y :",ytr.shape)
print("Test y :",yts.shape)
```

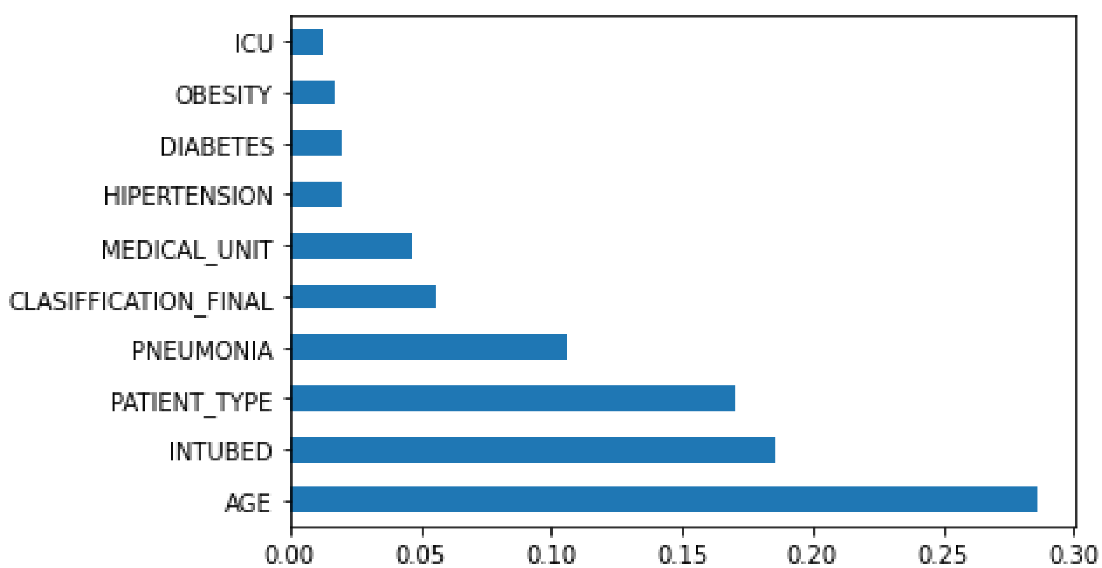
```
Train x : (413040, 20)
Test x : (103260, 20)
Train y : (413040,)
Test y : (103260,)
```

Now we proceed with the feature selection. We use an extra tree classifier to get the feature importance of each of the features and select the 10 most important features. We learnt about this method from this link in medium https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

[234]:
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

etc = ExtraTreesClassifier()
etc.fit(Xtr,ytr)
print(etc.feature_importances_) #use inbuilt class feature_importances of tree␣
 ↪based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(etc.feature_importances_, index=Xtr.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.00989687 0.04671262 0.         0.17069635 0.18607499 0.1056548
 0.285999   0.0028607  0.02020884 0.00873535 0.00613554 0.00871777
 0.02021294 0.01201643 0.01011423 0.01704982 0.01135474 0.00888319
 0.05568645 0.01298938]
```



[235]:
```
feature_list=feat_importances.nlargest(10)
feature_list
```

[235]:
```
AGE                     0.285999
INTUBED                 0.186075
PATIENT_TYPE            0.170696
PNEUMONIA               0.105655
CLASIFFICATION_FINAL    0.055686
MEDICAL_UNIT            0.046713
HIPERTENSION            0.020213
DIABETES                0.020209
OBESITY                 0.017050
ICU                     0.012989
dtype: float64
```

[236]:
```
fselect=['AGE','PATIENT_TYPE','INTUBED','CLASIFFICATION_FINAL','MEDICAL_UNIT','PNEUMONIA','ICU
X=df[fselect]
Y=df['DEATH']
Xtr, Xts, ytr, yts = train_test_split(X,y,test_size=0.2,random_state=42)
print("Train x :",Xtr.shape)
print("Test x :",Xts.shape)
print("Train y :",ytr.shape)
print("Test y :",yts.shape)
```

```
Train x : (413040, 10)
Test x : (103260, 10)
Train y : (413040,)
Test y : (103260,)
```

Now we run a logistic regression model after feature selection

[237]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(Xtr,ytr)
print("Logistic Regression Accuracy :",logreg.score(Xts, yts))
```

```
Logistic Regression Accuracy : 0.9620278907611853
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
```

Notice that the accuracy that we got for a simple logistic regression model with feature selection and handling of missing data is 3% more than the accuracy we got with a complex neural network without feature selection. This highlights how important data preprocessing is if you want higher accuracies. We will now see how the other models which we saw earlier without any data preprocessing behave now.

Decision Tree

[238]:
```
dt = DecisionTreeClassifier()
dt.fit(Xtr,ytr)
print("Decision Tree :",dt.score(Xts, yts))
```

```
Decision Tree : 0.9582316482665117
```

Random Forest

[239]:
```
rf = RandomForestClassifier()
rf.fit(Xtr,ytr)
print("Random forrest Accuracy :",rf.score(Xts, yts))
```

```
Random forrest Accuracy : 0.9588804958357544
```

Neural network: For our neural network since we are using only 10 features now we are going to have (2*10 - 1) = 19 neurons in each of the hidden layers

[240]:
```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation

nin = Xtr.shape[1]
nh = 19
nout = 1
model = Sequential()
model.add(Dense(units=nh, input_shape=(nin,), activation='relu', name='hidden'))
model.add(Dense(units=nh, activation='relu', name='hidden2'))
model.add(Dense(units=nout, activation='sigmoid', name='output'))
```

[241]:
```
from tensorflow.keras import optimizers

opt = optimizers.Adam(lr=0.001)
model.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.8/dist-
packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

[242]:
```
hist = model.fit(Xtr, ytr, epochs=30, batch_size=100, validation_data=(Xts,yts))
```

```
Epoch 1/30
4131/4131 [==============================] - 11s 2ms/step - loss: 0.1097 -
accuracy: 0.9574 - val_loss: 0.0934 - val_accuracy: 0.9601
Epoch 2/30
4131/4131 [==============================] - 10s 3ms/step - loss: 0.0931 -
accuracy: 0.9603 - val_loss: 0.0909 - val_accuracy: 0.9621
Epoch 3/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0921 -
accuracy: 0.9608 - val_loss: 0.0903 - val_accuracy: 0.9619
Epoch 4/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0915 -
accuracy: 0.9612 - val_loss: 0.0925 - val_accuracy: 0.9604
Epoch 5/30
4131/4131 [==============================] - 12s 3ms/step - loss: 0.0908 -
accuracy: 0.9613 - val_loss: 0.0894 - val_accuracy: 0.9623
Epoch 6/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0904 -
accuracy: 0.9615 - val_loss: 0.0893 - val_accuracy: 0.9626
Epoch 7/30
4131/4131 [==============================] - 10s 3ms/step - loss: 0.0901 -
accuracy: 0.9616 - val_loss: 0.0892 - val_accuracy: 0.9617
Epoch 8/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0898 -
accuracy: 0.9617 - val_loss: 0.0898 - val_accuracy: 0.9613
Epoch 9/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0896 -
accuracy: 0.9615 - val_loss: 0.0889 - val_accuracy: 0.9628
Epoch 10/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0895 -
accuracy: 0.9617 - val_loss: 0.0894 - val_accuracy: 0.9620
Epoch 11/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0892 -
accuracy: 0.9618 - val_loss: 0.0902 - val_accuracy: 0.9610
Epoch 12/30
4131/4131 [==============================] - 10s 3ms/step - loss: 0.0891 -
accuracy: 0.9618 - val_loss: 0.0900 - val_accuracy: 0.9620
Epoch 13/30
4131/4131 [==============================] - 10s 3ms/step - loss: 0.0890 -
accuracy: 0.9619 - val_loss: 0.0892 - val_accuracy: 0.9623
Epoch 14/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0890 -
accuracy: 0.9619 - val_loss: 0.0882 - val_accuracy: 0.9625
Epoch 15/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0888 -
accuracy: 0.9620 - val_loss: 0.0881 - val_accuracy: 0.9632
```

```
Epoch 16/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0887 -
accuracy: 0.9620 - val_loss: 0.0883 - val_accuracy: 0.9629
Epoch 17/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0887 -
accuracy: 0.9621 - val_loss: 0.0879 - val_accuracy: 0.9631
Epoch 18/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0886 -
accuracy: 0.9618 - val_loss: 0.0888 - val_accuracy: 0.9632
Epoch 19/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0884 -
accuracy: 0.9621 - val_loss: 0.0880 - val_accuracy: 0.9626
Epoch 20/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0884 -
accuracy: 0.9619 - val_loss: 0.0879 - val_accuracy: 0.9631
Epoch 21/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0883 -
accuracy: 0.9620 - val_loss: 0.0883 - val_accuracy: 0.9627
Epoch 22/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0882 -
accuracy: 0.9621 - val_loss: 0.0901 - val_accuracy: 0.9623
Epoch 23/30
4131/4131 [==============================] - 12s 3ms/step - loss: 0.0882 -
accuracy: 0.9621 - val_loss: 0.0887 - val_accuracy: 0.9626
Epoch 24/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0882 -
accuracy: 0.9623 - val_loss: 0.0883 - val_accuracy: 0.9628
Epoch 25/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0880 -
accuracy: 0.9622 - val_loss: 0.0882 - val_accuracy: 0.9628
Epoch 26/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0880 -
accuracy: 0.9621 - val_loss: 0.0882 - val_accuracy: 0.9625
Epoch 27/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0880 -
accuracy: 0.9622 - val_loss: 0.0885 - val_accuracy: 0.9627
Epoch 28/30
4131/4131 [==============================] - 11s 3ms/step - loss: 0.0881 -
accuracy: 0.9622 - val_loss: 0.0876 - val_accuracy: 0.9630
Epoch 29/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0879 -
accuracy: 0.9622 - val_loss: 0.0879 - val_accuracy: 0.9629
Epoch 30/30
4131/4131 [==============================] - 10s 2ms/step - loss: 0.0879 -
accuracy: 0.9622 - val_loss: 0.0895 - val_accuracy: 0.9628
```

[243]:
```
score, acc = model.evaluate(Xts, yts)
print('Test accuracy:', acc)
```

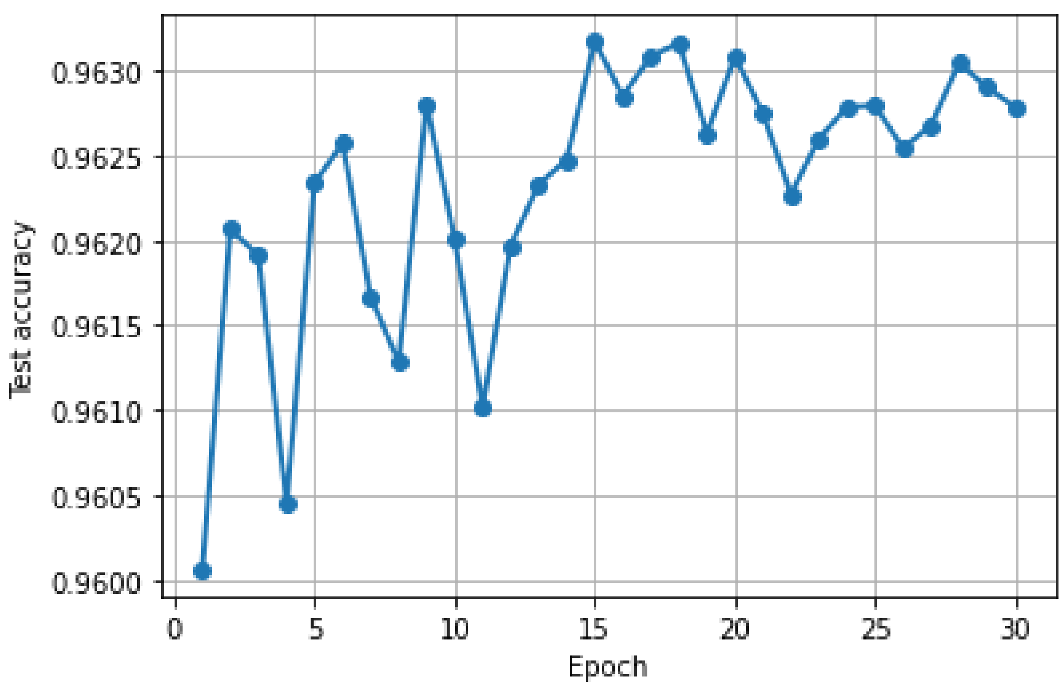
```
3227/3227 [==============================] - 5s 2ms/step - loss: 0.0895 -
accuracy: 0.9628
Test accuracy: 0.9627832770347595
```

We see that the neural network has the best accuracy with feature selection

We now plot the variation of test accuracy with each epoch.

[244]:
```
val_acc = hist.history['val_accuracy']
nepochs = len(val_acc)
plt.plot(np.arange(1,nepochs+1), val_acc, 'o-', linewidth=2)
plt.grid()
plt.xlabel('Epoch')
plt.ylabel('Test accuracy')
```

[244]:
```
Text(0, 0.5, 'Test accuracy')
```



We are also showing some other usefull metrics.

Confusion Matrix

[249]:
```
from sklearn import metrics
actual=yts
predicted=model.predict(Xts)
```

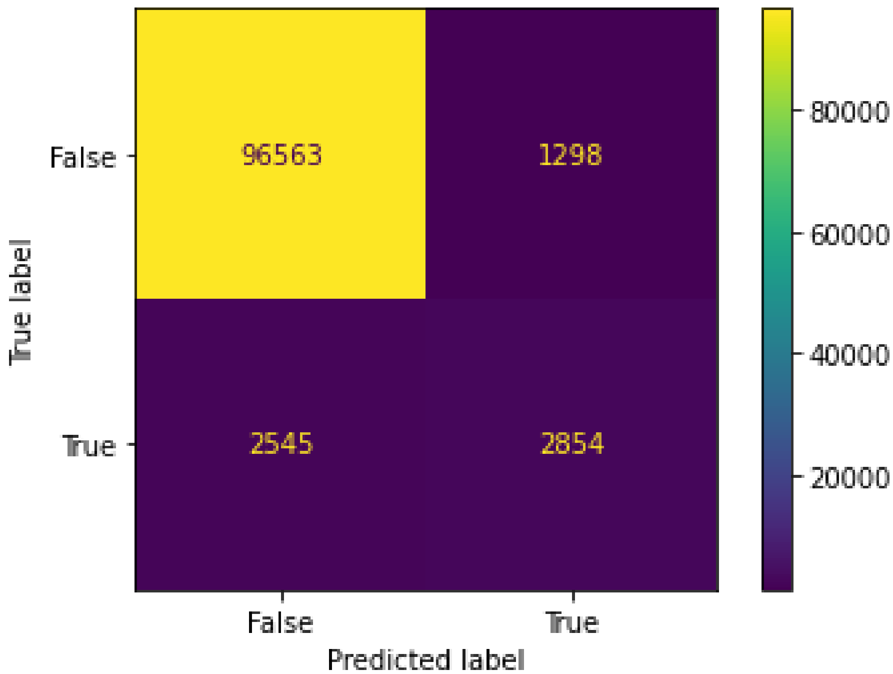
```
y_pred = np.round(predicted).tolist()
confusion_matrix = metrics.confusion_matrix(actual, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
 ↪confusion_matrix, display_labels = [False, True])

cm_display.plot()
```

3227/3227 [==============================] - 4s 1ms/step

[249]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fc0bfedcd30>



ROC Curves

[251]:
```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

[253]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(yts, predicted)
print('AUC: %.2f' % auc)
```

```
AUC: 0.97
```

[254]:

```
fpr, tpr, thresholds = roc_curve(yts, predicted)
```

[255]:

```
plot_roc_curve(fpr, tpr)
```