ML ASSIGNMENT 02

6037ps2021020

Dakheela Madanayake

Task 01

- Data cleaning process/preprocessing (if any)

1. Drop duplicates

```
In [158]: duplicate = nic.duplicated()
          print(duplicate.sum())

          4
```

```
In [159]: duplicate[duplicate]

Out[159]: 952     True
          1166    True
          1849    True
          3350    True
          dtype: bool
```

```
In [160]: nic[duplicate]

Out[160]:
```

	ID	Extracted_	Birth_year	Birthdayof_year	Serial_num	Check_digit	Special_ltr	Extracted1_isdigit
952	947942301V	947942301	94	794	230	1	V	True
1166	976161831V	976161831	97	616	183	1	V	True
1849	951461423V	951461423	95	146	142	3	V	True
3350	968583646V	968583646	96	858	364	6	V	True

```
In [161]: nic.drop_duplicates(keep='first', inplace =True)
```

```
In [162]: duplicate = nic.duplicated()
          print(duplicate.sum())

          0
```

2. Selected only numeric values present from the ID column and dropped the others.

```
In [149]: # nic.Extracted_1.isdigit()
          # whether only numeric value is present in the column of dataframe in Python
          nic['Extracted1_isdigit'] = list(map(lambda x: x.isdigit(), nic['Extracted_']))
          print (nic)

          # map(lambda x: x.isdigit(), nic['Extracted_1'])

                   ID Extracted_  Birth_year Birthdayof_year Serial_num Check_digit  \
0     911232910V  911232910          91             123        291           0
1     937370580V  937370580          93             737        058           0
2     937784210V  937784210          93             778        421           0
3     940491240V  940491240          94             049        124           0
4     942251610V  942251610          94             225        161           0
...          ...        ...         ...             ...        ...         ...
5018  988190969V  988190969          98             819        096           9
5019  988330809V  988330809          98             833        080           9
5020  988501069V  988501069          98             850        106           9
5021  995150549V  995150549          99             515        054           9
5022  995291649V  995291649          99             529        164           9

     Special_ltr  Extracted1_isdigit
0              V                True
1              V                True
2              V                True
3              V                True
4              V                True
```

```
nic = nic[nic.Extracted1_isdigit != False]
```

- Data transformation (if any)
  - Split the ID column to Birth year, Birthday of the year, Serial number, check digit and special character.

```
In [134]: nic['Extracted_']=nic['ID'].str[:9]
```

```
In [137]: nic['Birth_year']=nic['ID'].str.strip().str[0:2]
```

```
In [139]: nic['Birthdayof_year']=nic['ID'].str.strip().str[2:5]
```

```
In [141]: nic['Serial_num']=nic['ID'].str.strip().str[5:8]
```

```
In [143]: nic['Check_digit'] = nic['ID'].str.strip().str[8]
```

```
In [147]: nic['Special_ltr'] = nic['ID'].str.strip().str[9]
```

```
In [148]: nic.head()
```

Out[148]:

	ID	Extracted_	Birth_year	Birthdayof_year	Serial_num	Check_digit	Special_ltr
0	911232910V	911232910	91	123	291	0	V
1	937370580V	937370580	93	737	058	0	V
2	937784210V	937784210	93	778	421	0	V
3	940491240V	940491240	94	049	124	0	V
4	942251610V	942251610	94	225	161	0	V

```
In [145]: nic.Check_digit.unique()
Out[145]: array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], dtype=object)
```

- Data splitting

Now split the dataset into two sets using the **iloc** method (as X and y).

X should contain the features.

y should contain the class types.

Here we need a training data set to train the classification model as well as some test data to check the accuracy of the model build.
train_test_split function in sklearn.model_selection could be used for this purpose. The function accepts set of arrays as an input, split arrays or matrices input into random train and test subsets and returns them.

```
X = nic.iloc[:,2:5]
y = nic.iloc[:,5]
```

```
X.head()
```

	Birth_year	Birthdayof_year	Serial_num
0	91	123	291
1	93	737	58
2	93	778	421
3	94	49	124
4	94	225	161

```
X.shape
```

```
(5007, 3)
```

```
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Check_digit, dtype: int32
```

Train_test_split:

```
#training data set to train the classification model as well as some test data to check the accuracy of the model build.
(X_train, X_test, y_train, y_test) = train_test_split(X,y, test_size=0.22, random_state=42)
```

```
X_train.shape
```

```
(7324, 3)
```

```
X_test.shape
```

```
(2066, 3)
```

```
y_train.shape
```

```
(7324,)
```

```
y_test.shape
```

```
(2066,)
```

- Exact model with the hyper parameters.

After trying several models: SVM, KNN, XgBoost, RandomForest and Logistic Regression **KNN** model was chosen as the best model with the highest accuracy.

- How did you obtain the specific hyper parameters? Justify.

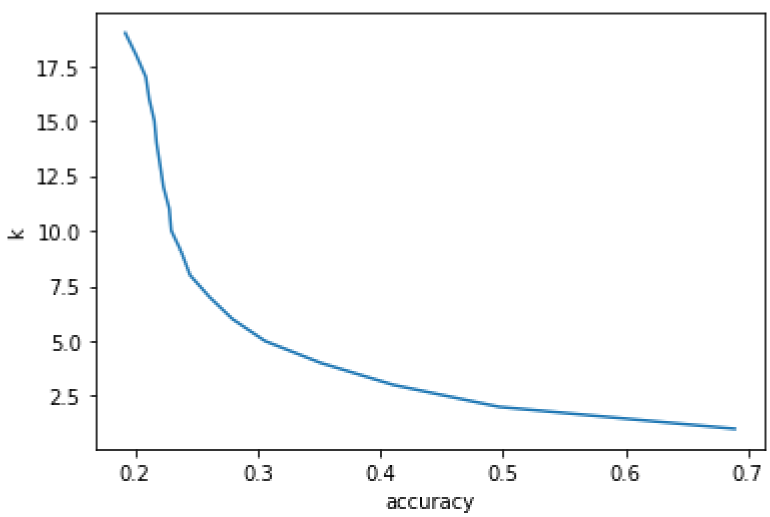
KNN model

1. Pick a value for K.
2. Search for the K observations in the training data that are "nearest" to the measurements of the unknown value.
3. Use the most popular response value from the K nearest neighbors as the predicted response value.

KNN ¶

```
ks = np.arange(1, 20)
scores = []
for k in ks:
    model = KNeighborsClassifier(n_neighbors=k)
    score = cross_val_score(model, X_train, y_train, cv=15)
    score.mean()
    scores.append(score.mean())
```

```
plt.plot(scores, ks)
plt.xlabel('accuracy')
plt.ylabel('k')
plt.show()
```



- **Training accuracy** rises as model complexity increases
- **Testing accuracy** penalizes models that are too complex or not complex enough
- For KNN models, complexity is determined by the **value of K** (lower value = more complex)

Hyper parameters tuning with grid search.

KNN hyper parameter tuning

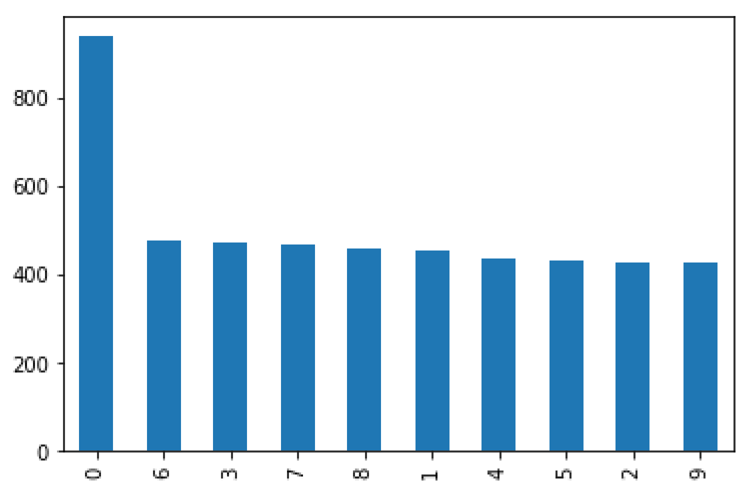
```
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,50))
p=[1,2]
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
#Fit the model
best_model = clf.fit(X_train, y_train)
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 26
Best p: 2
Best n_neighbors: 1
```

Other than the hyper parameters, the data set provided was an imbalanced dataset.

```
nic.Check_digit.value_counts().plot.bar()
```

```
<AxesSubplot:>
```



So I had to balance the data set for that I tried using sampling techniques such as random over sampling and random under sampling, SMOTE as well as tuning the class_weight parameter in the other algorithms used.

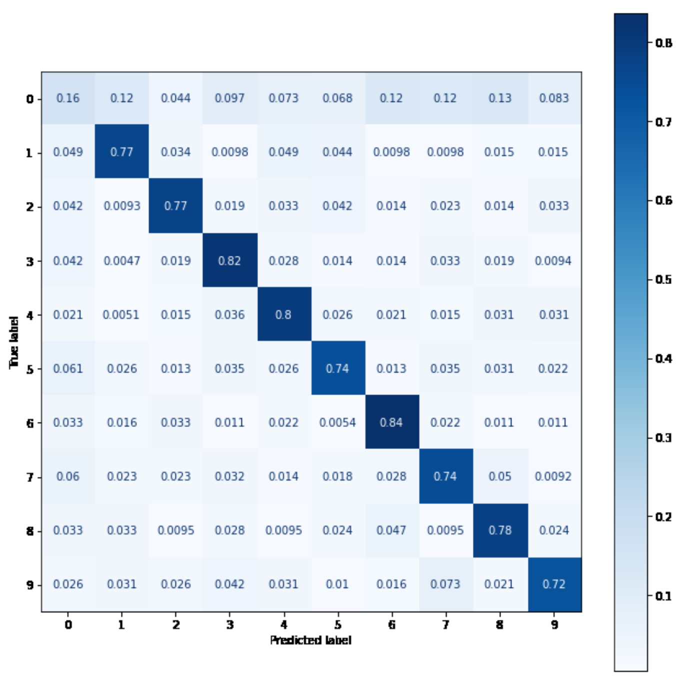
- Is the performance acceptable? If not, what could be the problem?
    - Using Hyper-parameters tuning can improve model performance by about 10% to a range of 71% for all evaluation matrices.
    - But even though the performance has improved at 71%, there are few misclassification of labels that we can improve.

```
              precision    recall  f1-score   support

           0       0.29      0.16      0.20       206
           1       0.74      0.77      0.75       205
           2       0.79      0.77      0.78       215
           3       0.73      0.82      0.77       212
           4       0.73      0.80      0.76       195
           5       0.76      0.74      0.75       228
           6       0.72      0.84      0.78       184
           7       0.70      0.74      0.72       218
           8       0.71      0.78      0.75       211
           9       0.74      0.72      0.73       192

    accuracy                           0.71      2066
   macro avg       0.69      0.71      0.70      2066
weighted avg       0.69      0.71      0.70      2066

0.712487899322362
```



Using the model the check digits of the following NIC numbers are calculated.

```
pred_test_knn
```

```
array([6, 0, 6, 4, 9])
```

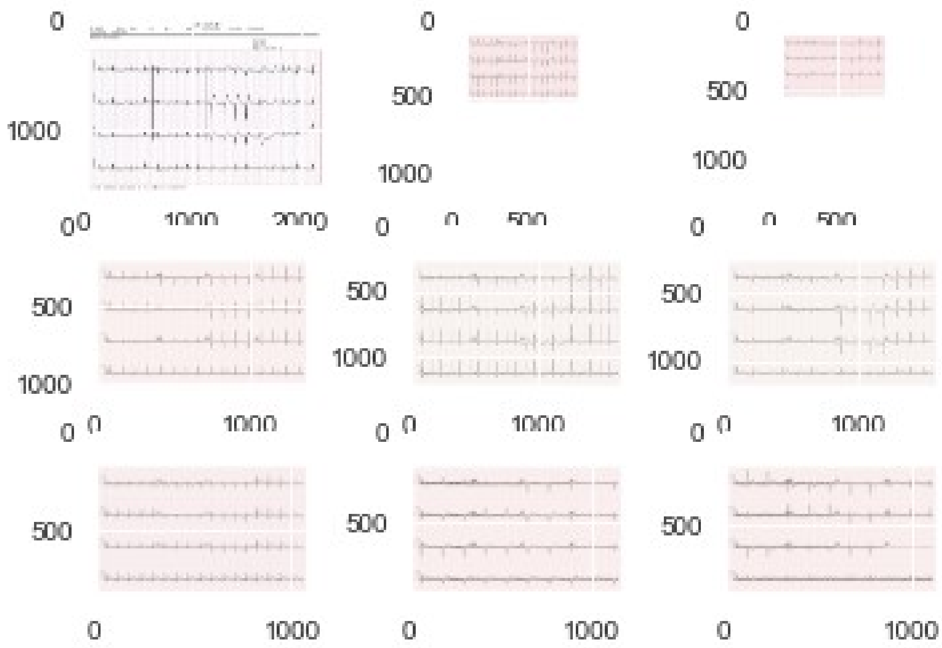
```
prediction_tsk1['Check_digit_knn'] = pred_test_knn
```

```
prediction_tsk1
```

	ID	Birth_year	Birthdayof_year	Serial_num	Extracted1_isdigit	Check_digit_knn
0	54178341	54	178	341	True	6
1	51782160	51	782	160	True	0
2	94693202	94	693	202	True	6
3	87352340	87	352	340	True	4
4	90705025	90	705	25	True	9

Task 2

- Data cleaning process/preprocessing (if any)
- Data transformation (if any)



  - The images of the dataset have some limitations. The images do not have sufficient resolution, and report image sizes are not standard.
  - Preprocessing the ECG images removing the background noise and pixels.

Two different strategies were identified to build the model,

1. To distinguish COVID-19 from No- Findings (that have normal ECG.

2. **Covid(+) vs Covid(-) - Here all 250 COVID-19 images, and equal amounts of images form the rest was selected.**

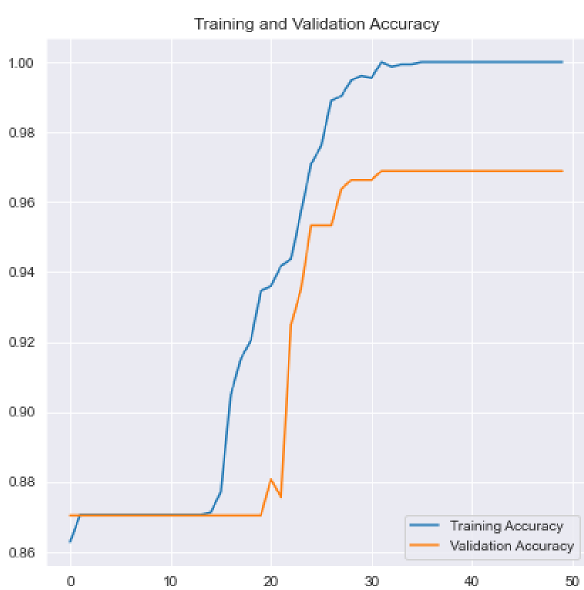
   The reason for choosing the equal amount of data in the classification process is to eliminate the imbalanced dataset effect. This method was carried out.

- Data splitting
  - Validation split = 0.25

- Exact model with the hyper parameters
    - CNN- Convolutional Neural Network.
    - AlexNet model, Rectified Linear Units

- How did you obtain the specific hyper parameters? Justify.
    - AlexNet used the Dropout method to overcome over-fitting.
    - Rectified Linear Units (ReLU) as the activation function to shorten the training time.
    - Adam Optimizer was used, because of its effective choice of hyper-parameters. T
    - The batch size is fine-tuned with parameter tuning. Different batch sizes have been tested in the training phase to achieve the least error rate, and the batch size optimized to 10.
    - Different learning rates were tested to ensure a lower error rate. Although decreasing the learning rate hyper-parameter slightly increased the training cost, it fine-tuned on 0.000001 to avoid local minimum. Epochs are tuned at to observe the robustness of the models.

- Is the performance acceptable? If not, what would you suggest for improving?
    - The performance of the model is okay but we can improve a little bit.



```
opt = Adam(learning_rate=0.000001)
model.compile(optimizer = opt , loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) , metrics = ['accuracy'])
```

```
history = model.fit(x_train,y_train,epochs = 50 , validation_data = (x_val, y_val))
```

```
Epoch 1/50
C:\Users\User\Anaconda3\lib\site-packages\keras\backend.py:4907: UserWarning: "`sparse_categorical_crossentropy` received `from
_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Wa
s this intended?"
  '"`sparse_categorical_crossentropy` received `from_logits=True`, but '
49/49 [==============================] - 441s 4s/step - loss: 0.4613 - accuracy: 0.8629 - val_loss: 0.3687 - val_accuracy: 0.87
05
Epoch 2/50
49/49 [==============================] - 134s 3s/step - loss: 0.3605 - accuracy: 0.8706 - val_loss: 0.3566 - val_accuracy: 0.87
05
Epoch 3/50
49/49 [==============================] - 140s 3s/step - loss: 0.3521 - accuracy: 0.8706 - val_loss: 0.3549 - val_accuracy: 0.87
05
Epoch 4/50
49/49 [==============================] - 136s 3s/step - loss: 0.3517 - accuracy: 0.8706 - val_loss: 0.3522 - val_accuracy: 0.87
05
Epoch 5/50
49/49 [==============================] - 132s 3s/step - loss: 0.3454 - accuracy: 0.8706 - val_loss: 0.3492 - val_accuracy: 0.87
05
Epoch 6/50
49/49 [==============================] - 132s 3s/step - loss: 0.3386 - accuracy: 0.8706 - val_loss: 0.3452 - val_accuracy: 0.87
05
```


Git Repository -
https://github.com/dakhz/6037ps2021020.git