Assignment 3.2 – ML development using Python

SIT 788

1. Introduction

The dataset "Acute Inflammations" [1], has been used for this case study. This data set has been prepared by a medical expert for presumptive diagnosis of two diseases of urinary system, namely: "*Inflammation of urinary bladder" and "Nephritis of renal pelvis origin".* These two diseases **constitute the two output class labels** for each feature row in this dataset. The features of this dataset include **6 variables**:

1. Temperature of patient [continuous floating-point values, ranging from 35C – 42C]

Apart from temperature, 5 categorical variables which take binary values [yes/no] are the remaining features of this dataset:

2. Occurrence of nausea
3. Lumbar pain
4. Urine pushing
5. Micturition pains
6. Burning of urethra, itch, swelling of urethra outlet

The dataset has a total of 119 datapoints, if we split the dataset on "*Inflammation of urinary bladder"* then the data **has 60 negative and 59 positive** class samples. Whereas, if we split the dataset on *"Nephritis of renal pelvis origin",* the data **has 69 negative and 50 positive** samples. The dataset is slightly imbalanced, also, due to two labels for each feature vector, this dataset is an example of multi-label dataset. This dataset requires supervised learning and makes the problem to be that of a multi-label classification. In the following sections, Decision Tree and Random Forest Classifier have been used to train a classifier and perform comparison across different metrics.

2. Creating ML Pipeline

The ML pipeline built for this consists of following steps:

1 – importing the required python packages

```
In [1]: #!pip install scikit-multilearn
        import pandas as pd
        import zipfile as zf
        import pickle
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        import codecs
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report
        from sklearn.metrics import multilabel_confusion_matrix
        from sklearn.metrics import accuracy_score
        from skmultilearn.problem_transform import LabelPowerset
        from time import time
```

Fig 1: imported libs

2 – Loading the data from source, the data is tab separated hence tab is used as the delimiter, also because the data is not in csv format by default therefore ***codecs.open()*** is used to read the data in

required pandas dataframe. The dataframe is then assigned meaningful column names denoting feature variable names.

Load the dataset

In [2]:
```
data_file = "./SIT788_3_2_Data/diagnosis.data"
udoc = codecs.open(data_file,'rU','UTF-16') #reading with "universal" type
df = pd.read_csv(udoc, sep='\t', decimal=',') #Reading tab seperated values and replacing ',' with '.'
df.columns =['temp', 'nausea', 'lumbar_pain', 'urine_pushing', 'micturian_pains',
             'urethra_pain', 'urinary_bladder_inflammation', 'renal_pelvis_nephritis'] #Add relevant column names
```

Fig 2: Loading dataset

3 – Exploratory data analysis and data preparation is done, as the dataset contains categorical strig values, hence they are transformed to numerical values as they are better suited for classification tasks (although in theory string categorical values can be used in case of decision trees, but sklearn does not offer support for such values].

Print 1st 5 rows and data tyes of column

In [4]:
```
display(df.head(5))
display(df.dtypes)
```

	temp	nausea	lumbar_pain	urine_pushing	micturian_pains	urethra_pain	urinary_bladder_inflammation	renal_pelvis_nephritis
0	35.9	no	no	yes	yes	yes	yes	no
1	35.9	no	yes	no	no	no	no	no
2	36.0	no	no	yes	yes	yes	yes	no
3	36.0	no	yes	no	no	no	no	no
4	36.0	no	yes	no	no	no	no	no

```
temp                           float64
nausea                          object
lumbar_pain                     object
urine_pushing                   object
micturian_pains                 object
urethra_pain                    object
urinary_bladder_inflammation    object
renal_pelvis_nephritis          object
dtype: object
```

Fig 3: Data pre-processing

Dataframe after data pre-processing:

Replace categorical string values wth numerical values

```
In [5]: df.replace(to_replace=['no', 'yes'], value=[0, 1], inplace=True)
        display(df.head(5))
        display(df.describe())
        display(df.dtypes)
```

	temp	nausea	lumbar_pain	urine_pushing	micturian_pains	urethra_pain	urinary_bladder_inflammation	renal_pelvis_nephritis
0	35.9	0	0	1	1	1	1	0
1	35.9	0	1	0	0	0	0	0
2	36.0	0	0	1	1	1	1	0
3	36.0	0	1	0	0	0	0	0
4	36.0	0	1	0	0	0	0	0

	temp	nausea	lumbar_pain	urine_pushing	micturian_pains	urethra_pain	urinary_bladder_inflammation	renal_pelvis_nephritis
count	119.000000	119.000000	119.000000	119.000000	119.000000	119.000000	119.000000	119.000000
mean	38.751261	0.243697	0.579832	0.672269	0.495798	0.420168	0.495798	0.420168
std	1.802346	0.431128	0.495673	0.471371	0.502096	0.495673	0.502096	0.495673
min	35.900000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	37.150000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	38.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
75%	40.600000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	41.500000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
temp                            float64
nausea                            int64
lumbar_pain                       int64
urine_pushing                     int64
micturian_pains                   int64
urethra_pain                      int64
urinary_bladder_inflammation      int64
renal_pelvis_nephritis            int64
dtype: object
```

Fig 4: Data visualization

4 – After pre-processing is done, the data is then split into training and testing set. The training set contains 60% of the data whereas 40% data belongs to the test set (Lower than this fraction in testing could not differentiate the two classifiers). The 6 feature variables belong to the feature vector X and the two decision variables constitute the output label vector Y. Random state is used for reproducibility of the results

Split data into train and test

```
In [6]: X, Y = df.loc[:, (df.columns != 'urinary_bladder_inflammation') & (df.columns != 'renal_pelvis_nephritis')], df[['renal_
        x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.40, random_state=42)
        print ("Data distribution")
        print(Y[Y==0].count())
        print(Y[Y==1].count())
```

```
Data distribution
renal_pelvis_nephritis          69
urinary_bladder_inflammation    60
dtype: int64
renal_pelvis_nephritis          50
urinary_bladder_inflammation    59
dtype: int64
```

Fig 5: Data splitting into train and testing set.

5 – The first model trained is **Decision Tree**, no hyper parameter tuning is done for any model and default parameters are used as they yield high accuracy for this dataset. The model is trained on training set and prediction is done on test set. Decision tree takes around 0.23 milliseconds for a single prediction and 5 milliseconds to train.

Training DecisionTree

In [7]:
```
clf_dt = LabelPowerset(DecisionTreeClassifier())

millis_a = int(time() * 1000)
clf_dt.fit(x_train, y_train)
millis_b = int(time() * 1000)
dif = millis_b - millis_a
print ("Decsion Tree train time: ", dif)

millis_a = int(time() * 1000)
y_pred_dt = clf_dt.predict(x_test)
millis_b = int(time() * 1000)
dif = millis_b - millis_a
avg_t = dif / len(x_test)
print ("Decsion Tree time per prediction: ", avg_t)
```

```
Decsion Tree train time:  5
Decsion Tree time per prediction:  0.22916666666666666
```

Fig 6: Decision Tree classifier

6 – Similarly, a second classifier: **Random Forest** is trained on the same dataset. As Random Forest is an example of ensemble classifier, and has more complexity to it, it takes slightly longer (around 0.69 milliseconds) to predict a single sample and 283 milliseconds to train.

Training Random Forest

In [8]:
```
clf_rf = LabelPowerset(RandomForestClassifier())

millis_a = int(time() * 1000)
clf_rf = clf_rf.fit(x_train, y_train)
millis_b = int(time() * 1000)
dif = millis_b - millis_a
print ("RF train time: ", dif)

millis_a = int(time() * 1000)
y_pred_rf = clf_rf.predict(x_test)
millis_b = int(time() * 1000)
dif = millis_b - millis_a
avg_t = dif / len(x_test)
print ("RF time per prediction: ", avg_t)
```

```
RF train time:  283
RF time per prediction:  0.6875
```

Fig 7: Random Forest classifier. *Note the increase in prediction and training time.*

The **confusion matrix** of both the classifiers [Fig 8], denote high values across all metrics. Both the classifiers are used with Label Powerset, this technique is used to transform a multi-label problem into a multi-class problem. This involves mapping combinations of all labels to a unique id which is treated as a label, this is what transforms a multi-label from to a multi-class scenario.

7 – Comparison of both models, both the models are compared across various metrics: accuracy, precision, recall and f1-score. Confusion matrix for multilabel classification is also used.

```
In [9]: print("########################Decision tree metrics################################")
        print (classification_report(y_test, y_pred_dt, zero_division=0))
        print("DT Accuracy: ", accuracy_score(y_test, y_pred_dt))
        print(multilabel_confusion_matrix(y_test, y_pred_dt))

        print("########################Random Forest metrics################################")
        print (classification_report(y_test, y_pred_rf, zero_division=0))
        print("RF Accuracy:", accuracy_score(y_test, y_pred_rf))
        print(multilabel_confusion_matrix(y_test, y_pred_rf))
```

```
########################Decision tree metrics################################
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      1.00      1.00        22

   micro avg       0.98      1.00      0.99        41
   macro avg       0.97      1.00      0.99        41
weighted avg       0.98      1.00      0.99        41
 samples avg       0.73      0.73      0.73        41

DT Accuracy:  0.9791666666666666
[[[28  1]
  [ 0 19]]

 [[26  0]
  [ 0 22]]]
########################Random Forest metrics################################
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        22

   micro avg       1.00      1.00      1.00        41
   macro avg       1.00      1.00      1.00        41
weighted avg       1.00      1.00      1.00        41
 samples avg       0.73      0.73      0.73        41

RF Accuracy: 1.0
[[[29  0]
  [ 0 19]]

 [[26  0]
  [ 0 22]]]
```

Fig 8: Confusion Matrix, Accuracy, Precision, Recall and f1-score.

The comparison shows that the Random Forest is the best performing model (discussed in detail in later section).

8 – Finally, the best model (Random Forest) is exported as a zip file.

Exporting and Saving zipped model

```
In [10]: model_f = open('best_model.pkl', 'wb')
         pickle.dump(clf_rf, model_f)
         model_f.close()
         zf.ZipFile('best_model.zip', mode = 'w').write('best_model.pkl')
```

Fig 9: Exporting the model

3. Results

The two models trained were:

Decision Tree: Decision Trees are a supervised learning algorithms which can be used for both classification and regression problems. Here, a decision tree classifier is used. While only being trained on 60% of the data, decision trees reach an **accuracy of 97.9%**, **98% precision and 100% recall** while taking only **0.23 milliseconds to predict and 5 milliseconds to train**. This shows the strength of decision trees as they can be used to create a base pipeline very quickly for benchmarking purposes. Another advantage of decision trees is that they provide insight into what features are important during model training and provides under the hood knowledge of the final model trained.

Random Forest: Random Forest come from the family of ensemble classifiers. Being a collection of multiple decision trees, they tend to be more robust to changes and require lesser data to generalize. Here, Random Forest reach a **100% test accuracy, precision and recall** while being trained on 60% data. This shows the strength of ensemble models over standalone models. Another advantage of Random Forest is that they are less prone to overfitting and hence can be used in real world scenarios. Because of the complexity and large number of trees involved, the training time is huge **(283 milliseconds)** as compared to Decision Tree.

4. Model Comparison and conclusion

On the given Dataset, the best model is Random Forest because of **higher accuracy and precision** than Decision Trees. Another key observation is, even when the size of training data is reduced, Random Forest maintain their higher accuracy and precision. The only downside of using Random Forest over Decision Tree is higher inference time, but as the difference is 0.4 milliseconds between both the prediction time, therefore the trade-off between time and higher accuracy-precision is acceptable. Also, due to the nature of the dataset being a medical dataset, accuracy-precision takes precedence over time as a misclassification is costlier than a delayed classification.

Moreover, the generalisability offers protection against overfitting scenarios which Decision Tree does not, this makes Random Forest the better performing model on this dataset.

Note: Both the classifiers reach **100% accuracy, precision, recall** once the **training size is increased to 70%,** even in this scenario the Random Forest would be the classifier to go with because of its robustness and ability to handle unseen data better, as the training data is not large in volume and the high metrics could be a result of overfitting.

References

[1] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.