Task2.1

March 20, 2023

1 Task 2.1P-Damon Vizl-s223545885- s223545885@deakin.edu.au

1.1 Question 1

The Data that I have used for this task is the University of Wisconsin Hospital Breast Cancer data. The problem that is to be solved in this task is to classify whether a person has a benign or malignant breast cancer. The data set is 699 rows long and has 9 distinct features and was recorded by the University of Wisconsin Hospital.
These features are; Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, and Mitoses. The class column is the target of the model with a 2 representing a benign growth and a 4 representing a malignant cancer.

The Machine learning model will need to take new data points with these 9 features and classify whether that patient has a benign or malignant growth. Due to the importance of the accuracy of this data it is critical that the model is highly accurate and has a very low return of false negatives.

1.2 Question 2

1.2.1 Data Ingestion

The code block below opens the csv file and using chardet.detect determines the encoding and decodes the csv file. We then add a columns names row and drop the "ID Number" column as it is not a feature of this data set.

[ ]:
```
import pandas as pd
import numpy as np
import chardet
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import pickle
import zipfile

#location of the diagnosis data
dataRefLoc = r"./breast-cancer-wisconsin.data"

#determine the encoding of the data so the data can be generic data.
```

```
def GetDataEncoding(url):
    with open(url, "rb") as f:
        data = f.read()
    result = chardet.detect(data)

    return result["encoding"]



#read the data into the data frame. As there is no header I have used the␣
 ↪argument header=None to stop the first row becoming the header
dataDF = pd.read_csv(dataRefLoc, delimiter= "\,", encoding =␣
 ↪GetDataEncoding(dataRefLoc), header = None, engine= "python")
#this line adds a new header row, data taken from the meta data of the data.
dataDF.columns= ["ID Number", "Clump Thickness", "Uniformity of Cell Size",␣
 ↪"Uniformity of Cell Shape", "Marginal Adhesion", "Single Epithelial Cell␣
 ↪Size", "Bare Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses",␣
 ↪"Class"]
#as the ID number has no bearing on the class (benign or malignant) we will␣
 ↪drop the first row.
dataDF.drop("ID Number", axis = 1, inplace= True)
```

1.2.2 Data Cleaning

As the data is raw without headers I have taken the attribute information from the data set website and added that as a header row for the data. I am also changing the commas to periods in the first column to better align with Australian data representation and processing of the data.

I run dataDF.infer_objects() to convert the objects to ints where possible. Then I run dtypes to confirm which columns need processing.

[ ]:
```
#infer better data types
dataDF = dataDF.infer_objects()
dataDF.dtypes
```

[ ]:
```
Clump Thickness               int64
Uniformity of Cell Size       int64
Uniformity of Cell Shape      int64
Marginal Adhesion             int64
Single Epithelial Cell Size   int64
Bare Nuclei                  object
Bland Chromatin               int64
Normal Nucleoli               int64
Mitoses                       int64
Class                         int64
dtype: object
```

We can see from infer_objects that all the data has been classified correctly except for the ‘bare

nuclei' column. I will force this column to numeric and then forward fill the NaN values.

```
[ ]: #takes the coloumn in question and forces it to be numeric.
dataDF["Bare Nuclei"] = pd.to_numeric(dataDF["Bare Nuclei"], errors="coerce")
#finds any NaN and replaces them with the row above using forward fill
dataDF["Bare Nuclei"] = dataDF["Bare Nuclei"].fillna(method="ffill")
#down cast to an int to reduce memory and increase processing.
dataDF["Bare Nuclei"] = pd.to_numeric(dataDF["Bare Nuclei"], downcast=␣
 ↪"integer")
dataDF.dtypes
```

```
[ ]: Clump Thickness               int64
Uniformity of Cell Size        int64
Uniformity of Cell Shape       int64
Marginal Adhesion              int64
Single Epithelial Cell Size    int64
Bare Nuclei                     int8
Bland Chromatin                int64
Normal Nucleoli                int64
Mitoses                        int64
Class                          int64
dtype: object
```

The code block below loads a LabelEncoder into the labelEncoder variable and then drops the "class" column (as this is our target) before encoding the data.

```
[ ]: labelEncoder = preprocessing.LabelEncoder()
targetName = "Class"
X = dataDF.drop(targetName, axis = 1).values
y = dataDF[targetName].values

for ij in range(0,X.shape[1]):
    X[:,ij] = labelEncoder.fit_transform(X[:,ij])
```

1.2.3 Training - Decision Tree

Now that we have ingested and cleaned the data it is time to train the data. I will first do a decision tree model.

```
[ ]: validation_size = 0.2 #the percentage of train to test data. 80 percent of the␣
 ↪data will be allocated to train and 20 to test

X_train, X_test, Y_train, Y_test =␣
 ↪train_test_split(X,y,test_size=validation_size)
DT = DecisionTreeClassifier(criterion="entropy", max_features=len(X[0]),␣
 ↪max_depth=10)

DT = DT.fit(X_train, Y_train)
```

```
y_prediction_DT = DT.predict(X_test)
```

1.2.4 Training - Random Forrest

Above we have trained the data through an Entropy Decision Tree. Below is a random forrest classifier test.

[ ]:
```
RFC = RandomForestClassifier(n_estimators=100)
RFC = RFC.fit(X_train, Y_train)
y_prediction_RFC = RFC.predict(X_test)
```

1.2.5 Evaluation of the Model

As this is a classification problem we can use the sklearn metrics within the classification report

[ ]:
```
#Below are two classification reports from the Sklearn package. This provides␣
 ↪precision, recall, f1-score and accuracy.
reportDT = classification_report(Y_test, y_prediction_DT)
reportRFC = classification_report(Y_test, y_prediction_RFC)
print(reportDT)
print(reportRFC)

#the confusion matrix provides us number of true positives, false positives,␣
 ↪true negatives and false negatives.
confusionMatrixDT = confusion_matrix(Y_test, y_prediction_DT)
confusionMatrixRFC = confusion_matrix(Y_test, y_prediction_RFC)
print(f"Decision Tree True Negatives: {confusionMatrixDT[0,0]}, False Negatives:
 ↪ {confusionMatrixDT[0,1]}, False Positives: {confusionMatrixDT[1,0]}, True␣
 ↪Positives: {confusionMatrixDT[1,1]}")
print(f"Random Forrest True Negatives: {confusionMatrixRFC[0,0]}, False␣
 ↪Negatives: {confusionMatrixRFC[0,1]}, False Positives:␣
 ↪{confusionMatrixRFC[1,0]}, True Positives: {confusionMatrixRFC[1,1]}")
```

```
              precision    recall  f1-score   support

           2       0.97      0.94      0.95        89
           4       0.91      0.94      0.92        51

    accuracy                           0.94       140
   macro avg       0.94      0.94      0.94       140
weighted avg       0.94      0.94      0.94       140

              precision    recall  f1-score   support

           2       0.98      0.98      0.98        89
           4       0.96      0.96      0.96        51

    accuracy                           0.97       140
```

```
   macro avg       0.97      0.97      0.97       140
weighted avg       0.97      0.97      0.97       140

Decision Tree True Negatives: 84, False Negatives: 5, False Positives: 3, True
Positives: 48
Random Forrest True Negatives: 87, False Negatives: 2, False Positives: 2, True
Positives: 49
```

1.3 Question 3

Accuracy Accuracy is a measure of total correct predictions divided by the total number of predictions. From our analysis on the accuracy we can see that the Decision tree had an accuracy of approximately 96% while the Random Forrest had an accuracy of 99%.
Precision Precision is calculated using the number of true positives divided by the total number of positive predictions. We can see that the Decision Tree was more precise in reporting true negatives and less precise in reporting true positives. The reverse is true for the Random Forrest. In the medical field it is more important to register true positives, this would provide weight in using this model as we are analysing breast cancer data. The DT had a weighted average precision of 96% and the RFC had a weighted average precision of 99%.
Recall Recall is a measure of how well a model can identify positive samples. It is calculated by dividing the number of true positives by the total number of positives. For the DT the recall had a weighted average of 96% and the RFC had a weighted recall of 99%.
F1 Score The F1 score is an overall indicator of the models performance. IT takes into account the precision and recall. The DT had a weighted average of 96% and the RFC has a weighted average of 99%. We can see that with an accuracy of 99% the RFC model is performing exceedingly well. When combined with radiographers and oncologists this would be an extremly useful tool in diagnosing cancer.

1.4 Question 4

We can see from the above metrics that the model that performs the best (most accurate) between the two is the Random Forrest model. This is due to the way that the model takes random samples and runs 100 tests to develop the model. As it is built on multiple Decision Trees it removes the accuracy issues presented by a single Decision Tree model. As our data set is relatively small the Random Forrest also performs better. Given the metrics mentioned above the RFC model outperforms the DT model in all regards. We know that the DT model is prone to variance in test sets and new data due to it's dependancy on it's training data. The RFC model rectifies this by selecting random allotments of data and random selection of features. This randomness uncouples the model from the training data more effectively.
Due to the RFC model running training on random features it removes issues with regard to the feature selection and any one feature that may be skewing the data is less relied upon.
Importantly the number of false negatives in the RFC model is 0. This is potentially the most important metric as a false negative could see a patient not undergo treatment or further investigation.

[ ]:
```
#Export the model
f = open("RFC_Model.pkl", "wb")
pickle.dump(RFC,f)
```

```
f.close()
```

Target Grade: HD

2 Reference

This breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.

@article{scikit-learn, title={Scikit-learn: Machine Learning in {P}ython}, author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E.}, journal={Journal of Machine Learning Research}, volume={12}, pages={2825–2830}, year={2011} }