Tumor Diagnosis using Machine Learning

Harry Akeroyd - N9997121 CAB320 - Artificial Intelligence Queensland University of Technology
May 29, 2020

Abstract

Artificial Intelligence is a constellation of technologies: Machine Learning (ML) to Natural Language Processing are forms of AI and herein ML and its application in diagnosing tumor conditions will be addressed. The system is developed using a provided database - $medical\_records.data$, sklearn libraries focusing on four types of classifier; *Nearest Neighbours, Decision Tree, Support Vector Machine* and *Neural Networks*. The data provided pertains to a patients medical test in which ten real-valued features are computed for each cell nucleus: per patient this has been repeated three times. The report and code demonstrates an efficient application of Machine Learning to assign the tumor diagnosis for a range of patients, with a lowest-accuracy score of 94.74%, produced by the Decision Tree and Nearest Neighbour Classifiers and a highest-accuracy score of 98.25% produced by the Support Vector Machine Classifier.

Keywords - Machine Learning, Classifier/s, k-NN: Nearest Neighbour, DT: Decision Tree, SVM: Support Vector Machine, NN: Neural Network, sklearn, Keras, numpy, cross validation, .

1 Functions

The following functions represent the work presented in $myQuack.py$: using the data provided by $medical\_records.data$.

1.1 Preprocessing the Dataset: $medical\_records.data$

The first integral step in developing an efficient and rigorous Machine Learning application is the preprocessing of the data. Preprocessing is the ability to change a dataset comprised of categorical and numerical information into data that a machine can parse. The data provided by $medical\_records.data$ is represented by the following table;

ID Number	Diagnosis	Radius	Texture	...	Fractal Dimension
842302	M	17.99	99	...	0.1471
842517	M	20.57	17.77	...	0.07071
84300903	M	19.69	21.51	...	0.2069
84348301	M	11.42	20.38	...	0.2597
84358402	M	20.29	14.34	...	0.1809

NB: '...' represents data points 2-9 and the ten-real valued features are provided for three cell nucleus's.

The aforementioned table is then preprocessed such that the cancerous diagnosis is separated and denoted as the *response variable* (Y) and is hot encoded under the following premise;

- Malignant = 1
- Benign = 0

The following data, excluding the patient ID is then assigned as the *feature variable* (X) and stored within a numpy array, as is Y. This preprocessing is developed in the function: $\textbf{\textit{prepare\_dataset}}(\textbf{\textit{dataset\_path}})$. In which the $dataset\_path$ represents $medical\_records.data$.

1.1.1 Standardizing the Data

Standard scaling is applied such that the column-data has a common scale. This is done using the $sklearn$ function *StandardScaler()*: this function normalises the features of the data. It does this be removing the mean and scaling to unit variance.

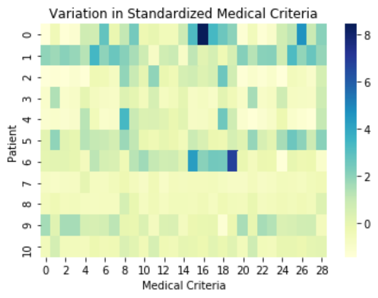

Figure 1: Standardized data for *medical_records.data*

1.2 Decision Tree Clasifier: DT

The first programmed machine learning application is a Decision Tree Classifier, represented and presented in the function ***build_DecisionTree_classifier***(***X_training, y_training***): see lines 100-200 in $myQuack.py$. A DT is a decision supported tool in which a tree-like graph is generated. Stemming from the root node, which generally represents the entire population of data, is a internal node which is arrived by splitting or can be pruned, extending from the internal nodes can be leaf nodes which are the full extension of an internal node and cannot be expanded upon. Utilising the sklearn library's allows for a majority of the 'heavy lifting' to be executed through the *tree.DecisionTreeClassifier* function. Furthermore, the optimization of the hyperparameter is completed through the use of a cross validation approach *GridSearchCV*: note this optimization is completed for all classifiers. Tuning of this hyperparameter is ran through a grid-search approach and is done so to find the optimal cross validation score. For the purpose of this assessment these hyperparameters are predetermined and therefore a single parameter and the estimator object are passed to *GridSearchCV*. The hyperparameter selected for the DT is *max depth*. This is the maximum depth in which the tree will split. It is to be noted that on a 32-bit machine the maximum depth value is 30. The following code represents the pseudo-code for the DT Classifier;

```
Data: Training data, split into feature and response variables
Result: DT Classifier
  1. Initialise the classifier from the sklearn library;
  2. Execute the grid search for hyperparameter optimization;
  3. Fit the data to the estimator;
  4. Re-train the estimator using optimized parameter;
  5. Re-fit the data to the estimator.
```

Algorithm 1: Decision Tree Classifier Pseudo-Code.

Please note the above pseudo-code is relevant for the ensuing classifiers with the respective sklearn and hyperparameters to be used.

1.3 Nearest Neighbour Classifier: k-NN

A k-NN Classifier has been presented in ***build_NearrestNeighbours_classifier***(***X_training, y_training***): corresponding lines 100-200 in $myQuack.py$. A k-NN Classifier is a instance-based learning algorithm with the understanding that it assumes that similar things exits in close proximity to each other. Refer to *Section 1.2 Decision Tree Classifier: DT* for the material on hyperparameter optimization: substitute *n_neighbours* for the optimized parameter. The implication of increasing or decreasing the number of neighbours, *n_neighbours*, has two significant implications;

1. *Increasing*: Stability is able to be achieved due to majority voting/ averaging. This is suitable up until the K value has exceeded its viable value and produces errors.
2. *Decreasing*: In-stability is met as K decreases. In the case that K=1 a query point surrounded by several red dots and a single green would predict the green dot as it may be the single nearest neighbour.
3. *Majority Vote*: in the case of this instance the selected K is generically an odd number, allowing for a tiebreaker. For this computation a query point is assigned to the data class which has the most representatives within the *n_neighbours* of the point.

The pseudo-code for the development of the k-NN can be seen in *Algorithm 1*, where the following will be supplemented; the hyperparameter for optimization is *n_neighbour* and the classifier is *neighbours.KNeighboursClassifier()*.

1.4 Support Vector Machine Classifier: SVM

The third classifier developed for this assessment is the Support Vector Machine Classifier, denoted by SVM and presented in ***build_SupportVectorMachine_classifier***(***X_training, y_training***), this can be found through lines 100-200 in $myQuack.py$. A SVM is an approach in which each item of data is plotted in an n-th dimensional space (with n denoting the number of features derived from the data, in this instance this value is 30). The value of each feature is represented on this plot and corresponds to a specific coordinate. Classification is then achieved by finding the hyper-plane that correctly differentiates the two classes. The hyperparameter tuned through *GridSearchCV* for this SVM is the 'C' parameter. This parameter tells the SVM optimization the degree in which misclassification is to be avoided in the training set. For larger values assigned to C, optimization will be selected upon a smaller-margin hyperplane. Inversely, for smaller values assigned to C the optimizer will seek a larger-margin separating the hyperplane, independent of the amount of misclassifying points. The pseudo-code for the development of the SVM can be seen in *Algorithm 1*, where the following will be supplemented; the hyperparameter for optimization is *C* and the classifier is *svm.SVC()*.

1.5 Neural Network: NN

The final developed classifier is a Neural Network, denoted by NN and represented in the function ***build_NeuralNetwork_classifier***(***X_training, y_training***): see *myQuack.py* and refer to lines 100-200. A NN, which can be refereed to as a connectionist system is a systems that is loosely modelled around the workings of a biological neural network (brain). The aforementioned function utilises a Multi-layer Perceptron (MLP) NN: a supervised learning algorithm trained on a dataset. Through classification a MLP can learn a non-linear approximation when given a set of features and a target, referred to prior as feature and response variables. An MLP is different from a logistic regression, from the input to the output layer a number of non-linear layers can be present, these are called hidden layers and are the focus of optimization. These hidden layers represent the *ith* element which in turn represents the number of neurons in the *ith* hidden layer. The pseudo-code for the development of the NN can be seen in *Algorithm 1*, where the following will be supplemented; the hyperparameter for optimization is *hidden_layer_sizes* and the classifier is *neural_network.MLPClassifier()*.

1.6 Executing Function

The executing function is used to display the results of the classifier. Each classifier is assigned the variable *clf*. The four classifier are initially commented out such that the desired classifier results can be displayed by simply removing the comment on the variable *clf*. The display of individual results are uniform and easily repeatable. The format for the aforementioned can be seen in Figure *Sections 2.1 ... 2.4*. The following list represents the results presented and their purpose in evaluating the performance of a classifier;

1. *Classifier Accuracy*: Numerical value to represent the accuracy of the classifier.
2. *Confusion Matrix*: Used to represent the *True* classifications against the *predicted* for each response type.
3. *Precision Score*: A value produced from the division of relevant instances of classification against the retrieved instances.
4. *Recall Score*: Also referred to as the sensitivity of the classifier, is a value produced from the division of the sum of relevant instances that were truly retrieved.
5. *Cross-Validation Score*: Estimates the expected accuracy of the selected model on out-of-training data.

1.6.1 Cross Validation

It is also pertinent to address cross validation as this method was used for hyperparameter optimization for each classifier. More preceisly, this was done through the use of *GridSearchCV* and the pre-selected hyperparameter, mentioned in *Sections 1.2 ... 1.5*. When applying cross validation the data is shuffled and done so to prevent any unintentional ordering errors and split into k folds. K models are then fitted to f of the data (denoted as the training split) and evaluated against 1 of the data (denoted as the test split). The resultant of each evaluation is averaged together for a final score. The final model is then fitted on the entire dataset for operationalization.

2 Reporting of Results and Analysis

The following section represents the performance characteristics for the code. Each classifier is individually tested and the results reported.

2.1 Performance Reporting: Decision Tree Classifier

The data represents the performance of the Decision Tree Classifier. The classifier correctly identified the tumor diagnosis with an accuracy of **94.736842**.

		True Diagnosis		
		Malignant	Benign	Total
Screening test	Malignant	68	3	71
	Benign	3	40	43
	Total	71	43	114

- *Precision Score*: 93.023256%
- *Recall Score*: 93.023256%
- *Cross-Validation Score*: [0.95652174, 1.0, 0.86956522, 0.91304348, 0.86363636]

2.2 Performance Reporting: Nearest Neighbour Classifier

The data represents the performance of the Nearest Neighbour Classifier. The classifier correctly identified the tumor diagnosis with an accuracy of **94.736842**.

		True Diagnosis		
		Malignant	Benign	Total
Screening test	Malignant	68	3	71
	Benign	3	40	43
	Total	71	43	114

- *Precision Score*: 93.023256%
- *Recall Score*: 93.023256%
- *Cross-Validation Score*: [0.95652174, 1.0, 0.86956522, 0.91304348, 0.86363636]

2.3 Performance Reporting: Support Vector Machine Classifier

The data represents the performance of the Support Vector Machine Classifier. The classifier correctly identified the tumor diagnosis with an accuracy of **98.245614**.

		True Diagnosis		
		Malignant	Benign	Total
Screening test	Malignant	71	0	71
	Benign	2	41	43
	Total	73	41	114

- *Precision Score*: 100.0%
- *Recall Score*: 95.348837%
- *Cross-Validation Score*: [0.95652174, 0.95652174, 0.91304348, 0.91304348, 1.0]

2.4 Performance Reporting: Neural Network

The data represents the performance of the Neural Network Classifier. The classifier correctly identified the tumor diagnosis with an accuracy of **97.368421%**.

		True Diagnosis		
		Malignant	Benign	Total
Screening test	Malignant	69	2	71
	Benign	1	42	43
	Total	70	44	114

- *Precision Score*: 95.454545%
- *Recall Score*: 97.674419%
- *Cross-Validation Score*: [0.95652174, 0.91304348, 0.95652174, 0.95652174, 0.90909091]

3 Conclusion

The report has shown the use and implementation of Machine Learning Algorithms and their application in predicting tumor diagnosis's for patients based on a medical data collected. This was done and applied to four classifiers; Decision Tree, Nearest Neighbour, Support Vector Machine and a Neural Network. Before this classifiers were initialised the data was preprocessed. This allowed for standardizing of the data to occur and feature and response variables to be extracted. The results of the program demonstrated that the most effective classifier was the Support Vector Machine Classifier with an accuracy score of 98.25% and the least accurate classifiers were the Decision Tree and Nearest Neighbour Classifiers with a score of 94.74%.