Classifying Electrocardiogram Recordings

CSM6420 - mah60 - Matthew Howard

May 2019

Contents

1 Observations and Preprocessing

The following report will discuss the development of classification models using machine learning algorithms to accurately predict electrocardiogram (ECG) recording types. This is done from two sets of data that were given to us to help train and build the individual models. This section of the report will discuss the observations and preprocessing that was done on the data given to us to create a suitable structure for the different models to be built on.

The data for this project is from PhysioNet Computing in Cardiology Challenge 2017, it is the ECG short recordings for detecting cardiac arrhythmia (irregular heartbeat).

1.1 Observations

There are two sets of data that were given to be processed, containing around 13,062 recordings for the training data and 4,000 recordings for the test data. The different types of datasets are the features and signal data for testing and training purposes. Each of these csv files have an ID column which represents the record being tested. Then in the training data the ECG segment types are given in the Type column, these are;

- N – Normal sinus rhythm
- A - Atrial fibrillation (AF)
- O – Other cardiac rhythm
- ˜ - Noise Segment

These types therefore help to classify the different ECG segments. The observations that were made on these datasets were;

ECG feature dataset;

- The different features were developed from the ECG waveform using various signal processing algorithms. There are no specified algorithms for the processing.
- There are in total 187 different features labelled as ‘F1’, ‘F2’, etc.
- There are some items that are clearly misclassified. Such as ‘F158’, ‘F170’ and ‘F171’ and some of the ECG records in the training dataset as they are constantly NaN or zero in value. There are also other features that could be determined to be misclassified. This is due to most of the feature values being zero with few having an actual value.

ECG signal dataset;

- There are 6,000 points that can be used to plot the wavelength of ECG segment. These columns are labelled as ‘X1’,’X2’, etc.
- The segment recordings are 20 seconds long and are at a frequency of 300 hertz.
- There are some recordings that were cut short before the end of the 20 seconds count.

The observations made against the dataset can be used to develop a more suitable dataset for the classification model that will be built.

1.2 Preprocessing

As we have now observed how the data sets are structured, we can now preprocess the data from the different models. The preprocessing done can be seen in the following sub chapters.

1.2.1 Features Preprocessing

The observations made about the feature dataset is that the data will need to be rebuilt to fit our needs and fix the current errors in the dataset. This can be done using dimensionality reduction techniques and other methods that are described below.

- Split the records into X and y values - this is done to help build the models as X defines how the class will look in that example and y is its resulting class, the ECG segment type.
- Remove misclassified features – This will remove the features that were observed to be misclassified within the dataset. This is done to remove the possibility that they could become a deciding factor of what class the ECG segment is.
- Replace null records – This will be to remove/replace null records with a mean value of that feature. This is done to make sure that all the records have data within them and will therefore help in the classification training process as dummy data. However, this could lead to some items being misclassified.
- Dimensionality reduction - a method that applies an algorithm against each feature and scores each feature. Once scoring is complete the top K items are chosen. This is done to reduce overfitting within the classification model, as if the model is not fully dependent on each feature the more leeway the model will have to unexpected data.

1.2.2 Signal Preprocessing

The observations made about the signal dataset tells us that the following preprocessing is required to create a complete dataset that could be used for creating a model with;

- Normalising signals 20 seconds long – This will be done by locating the null values at the end of the records and replacing them with the most frequent value. The reasoning behind this is to reduce the possibility of error or the classifier modelling itself to expect values less than 20 seconds and causing error in the final classification.
- Types to Categorical Values – This is where the types are turned to integer values and then to categorical structure. This is done to ensure a neural network can distinguish between classes. The weighting in the neurons will decide which class it represents.
- Reshape features to 3D array – This will transform the shape of the original data to [batch size, time steps, input dim]. This is done to ensure that the neural network understands that there is only one time step within the data. This can then be used for neural network models like Long Short Term Memory (LSTM).

2 Classifiers

After the observations and preprocessing have been reviewed, the chosen classifiers used to build the models for classification of ECG type are Random Forest (RF) and Artificial Neural Networks (ANN). The following chapters will expand on these classifiers and why they were chosen.

2.1 Random Forest

Random Forest takes advantage of the decision tree (DT) classifier and enhances it to reduce overfitting and can handle multiple features. To understand RF and how it enhances DT look at figure 1 and look at the following points;

- Decision Trees (DT) - DT can be modeled to look like an upside down tree. The top of the model represents the roots as the root node, then each node calculates the entropy value from the data given to it. This then leads down the branch to the corresponding value and will reach the next node. This will continue till there are no more nodes to branch off to, this will be the leaf node that will decide the final classified result..
- Random Forest (RF) - RF takes these DT and creates a forest from them. Therefore creating x number of estimators/trees, these estimators are built using random features selected from the training data.
- Majority Voting - RF decides the results based on majority vote, each estimator is given the test data and then they calculate a result. These results are collected and the highest frequency of a class is the outputted result.

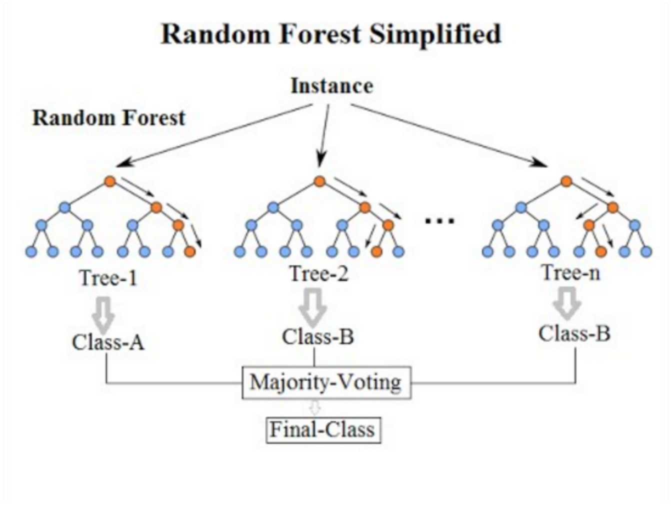

Figure 1: Random Forest Diagram[1]

The reasoning for choosing RF over another classification models, was due to its high durability to overfitting, as each tree is given a random selection of features. Then the main advantage to RF is that you can have multiple features with a large amount of training data[2], this is due to its random feature selection for each tree. Therefore with the 187 features that are contained within the datasets, the model should easily be able to develop a successful classification model from the training data given. The final reason for picking a RF algorithm is due to its parameters being simplistic and intuitive, therefore picking the best hyperparameters will become easier.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) were developed from the structure of a neurons network system inside the brain. The structure of a neuron network has three layers; input, hidden and output layers. These layers have their own formulas that calculate the weightings between the set amount of neurons in the layers. The weights can be represented as branches between the neurons, these weights help to form the results that are produced by the output. Once complete the neurons weights will be returned, these weights can be used to calculate what class the input data was given, as they represent categorical values. Hence preprocessing y to be categorical, where [0,0,1,0] will translate to the value 2. The final part to a Neural Network is that there is a parameter which is called 'Epochs', this is the number of training cycles. During each training cycle the current neuron inherits its predecessors weighting. A diagram representing an ANN can be seen in figure 2.

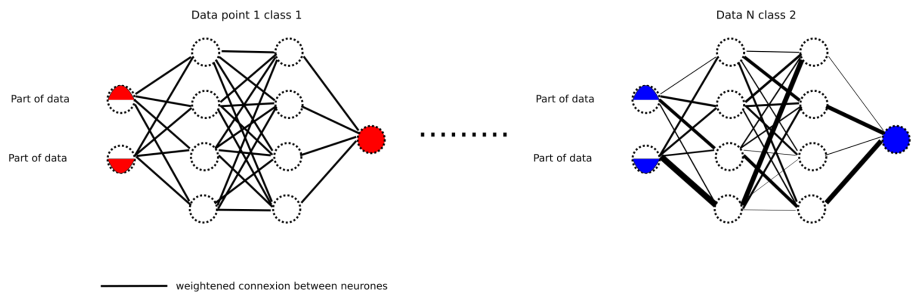

Figure 2: Artificial Nueral Network[2]

The strengths of the ANN are that some very complex models can be trained, that can do a numerous amount of things to get a final output. This is beneficial to our ECG segment classification as this allows for flexibility on how complex the model is and what we want to do with the data between the neurons. Furthermore, the flexibility that ANN range around could be even greater, this is due to the multitude of layers that could be used to generate different models. These layers will apply different calculations to the weighting between neurons and as a whole will change the final result. Layers that may benefit the ECG segment classification are;

- Dense - represents a standard layer within a neuron network, where given formulas can change the weighting to the neurons.
- LSTM - This layer stands for long short term memory, the layer applies the two activation functions 'tanh' and 'sigmoid'. The layer also chooses what values should be forgotten or remembered. Depending on their impact on the overall model.
- DropOut - this will reset at random a certain percentage of neuron weights in the given layer. This is used to reduce underfitting and overfitting and helps the ANN to learn more from the data given and not memorize the resulting weighting values.

Additionally, there are multiple other hyperparameters that can be altered within the layers, compiling stage and the model itself to improve the overall accuracy. On the other hand, this can also be a weakness towards choosing ANN due to the complexity of the structures. Therefore, Building and training an efficient ANN takes a lot of time, resources and a deep understanding of how the different models work, development becomes more of an art from than anything else. In spite of that, once a model is produced correctly they are extremely strong at what they were designed to do. In this case classify ECG segments.

3 Building Classifiers

The decided classification modifiers that were chosen were Random Forest (RF) and Artificial Neural Networks (ANN). The next stage to making the final classifiers was to build the models architecture using hyperparameters and other decisions, such as on how to structure the models. This will be discussed further in the following chapters.

3.1 Random Forest

The architecture for the RF classifier was designed to be simplistic, due to the low amount of hyperparameters being used. The decision on the best hyperparameters was done through grid search and multiple tests. The parameters that were chosen to define the structure of the RF were[8];

- n_estimators - This is the amount of DT inside the RF.
- max_features - The function that decides the max amount of features to be split among the trees.
- max_depth - The maximum amount depth of each tree, until the nodes are cut off and turned into leafs, this is known as pruning.
- min_samples_split - The minimum amount of samples used to split an internal node within the DT.
- min_samples_leaf - The minimum amount of samples required to be counted as a leaf node.
- bootstrap - If bootstrap samples are used.
- Dimensional Reduction Size - What size the number of features will be reduced to.

These parameters decide the overall structure of the model and how they will perform. There are other hyperparameters that also effect the final precision, however, the decision was to focus on the selected parameters above.

The Dimensional Reduction size decides on how many features are selected from the original amount using a feature selection algorithm called 'Sklearn SelectKBest'[9] that scores the value of each feature and picks the top K features to be given. Therefore this method was used as it reduces the chance of overfitting as described in the preprocessing chapter.

The method that was used to test the overall effectiveness of the RF model in classifying the ECG segments was to use grid search to select the best parameters, this takes a dictionary of parameters and will test each combination of hyperparameters. Once complete the output from the grid search will be the parameters that score the highest accuracy. The function used was the 'Sklearn RandomizedSearchCV'[10], this method also performed cross validation against each build. The cross validation method used was K fold validation and was used to access overall accuracy of the model and to discover whether the model has been overfitted. This is due to the model being trained and assessed K times, if the accuracy of the model differs too much or shows bias, this can show that the model is overfitted and needs to be remodeled. On top of this, the data given to the function is split to create training data and test data from the original source, each time selecting a different dataset from the orginal. Meaning the overall model is evaluated on different sources.

Overall, these two methods support the final RF architecture being built with the optimal hyperparameters for the ECG classification. This is due to the grid search and cross validation.

3.2 Artificial Neural Networks

Creating an efficient Artificial Neural Network (ANN) is extremely complex, due to the amount of options that can be configured. This can be seen from the different models designs to the amount of hidden layers within the ANN. The main methods that could be used to assess the ANN are;

- Grid search - As described in the RF section of this chapter, this will allow for the model to choose optimal parameters. However, this only works for the training layer of the ANN and will therefore only effect the Epochs and Batch Size. Meaning, another method will need to be used to choose the most optimal architecture for the current ANN model. Therefore we will have to use other methods to assess the structure of the model and its parameters.
- Cross Validation - Explained in the RF section of this chapter.
- Testing and evaluating the models - Changing the values within the structure of the ANN or the structure of the layers and their corresponding values. For each attempt evaluating the overall score and error rate of the model.

Overall, the best way to build the architecture of the model is to use the Grid search and Cross validation to assess the effectiveness of the current model and this will choose the optimal parameters for the outer layer of the ANN, when being trained. Then to improve the inner structure of the ANN the main method to use would be to alter the different values within the layers and modify the layers structure inside the different models by removing and inserting layers from the hidden layer structure or input layer from the overall model and assessing their performance. This optimal architecture can also take into account the different models that could be built for example LSTM and CNN models[3]. This method of choosing the best architecture for the job will be done through research into different models and experimentation to see what values will work to get the best overall result, this will therefore take a lot of time and resources to complete.

4 Experimentation And Results

Through the previous chapters we have discussed and concluded which preprocessing techniques and classifiers will be used, also how to thoroughly evaluate the results and improve the classifiers. The final stage of this project is to build and train the selected classifiers for the ECG segments to understand what Type the ECG segments are. The following chapters will discuss the experimentation done with the classifiers architecture and results of the individual models, then these classifiers will be concluded and evaluated.

4.1 Random Forest

The experiments that were performed against the Random Forest model can be seen in Table 1. The table explains the parameters that were used to gain the accuracy of that specific model. Some of the parameters were decided upon using a grid search algorithm, as explained in the chapter Building Classifiers - Random Forest.

The first three experiments recorded were the tests that made sure that the algorithm worked as planned and how these experiments were performed before any preprocessing was used. Therefore, these results can be used a baseline to evaluate how the preprocessing and the other methods that were used, these were done to improve the efficiency of the results. The highest accuracy that was achieved by the first three tests was 0.80. Compared to the rest of the tests ranging from ID 4-10, there was a significant improvement on the overall performance of the RF model, hitting the highest score of 0.86 accuracy.

The preproccessing that was applied to the model and its dataset was that all the features that were deemed to be misclassified were removed. As well, that the missing records were given the most frequent values of their class. The preproccessing obviously improved the overall accuacy of the model to a point were the accuracy increased over 0.80.

The overall effectiveness of model was improved by the grid search method, as can be seen by test 4 reaching a accuracy of 0.85. The grid search helped to find the hyperparameters that would improve the accuracy of the model. After this method of grid search was complete learning what variables best improved the model became easier.

The tests to decide what the best model to submit to the Kaggle competition were based on the results from test 4, which showed the highest accuracy. The tests that sought to improve test 4 were test IDs; 8,9 and 10. The main hyperparameter that was altered during these tests was the dimensionality reduction/feature selection. This was due to the dimensionality reduction not being part of the grid search being used. Overall, test 9 came out victorious with a score of 0.86 accuracy, this test had no dimensionality reduction, showing that the model did not need the amount of features reduced to improve it and this preproccessing technique was actually hindering the models capability. This could be due to where the RF classifier selects features from each tree at random, therefore meaning more features can equal less of a bias in the RF model. Test 9 on account of having the highest accuracy of the models was used as the Kaggle classifier model and achieved a score of 0.8175 overall from the test data given for the competition.

Test ID	Accuracy	n_estimators	max_features	max_depth	min_sample_split
1	0.76	100	Default	2	Default
2	0.76	100	Default	2	Default
3	0.80	100	Default	5	Default
4	0.85	350	auto	70	2
5	0.83	350	auto	70	2
6	0.84	300	auto	50	2
7	0.84	300	auto	50	2
8	0.84	350	auto	70	2
9	0.86	350	auto	70	2
10	0.84	350	auto	70	2

Test ID	min_sample_leaf	Bootstrap	Dimensionality Reduction Size	Notes
1	Default	Default	None	No preprocessing
2	Default	Default	None	No preprocessing
3	Default	Default	None	No preprocessing
4	1	False	100	Removed miss classified features, uses grid search
5	1	False	100	Removed miss classified features, Test with optimal parameters from Test 4
6	1	False	120	Removed miss classified features, uses grid search
7	1	False	None	Removed miss classified features, uses grid search
8	1	False	100	Removed miss classified features, Test with optimal parameters from Test 4
9	1	False	None	Removed miss classified features, Test with optimal parameters from Test 4
10	1	False	120	Removed miss classified features, Test with optimal parameters from Test 4

Table 1: Random Forest Results

4.2 Artificial Neural Networks

Overall, the experimentation that was performed to try and build a working model that would efficiently classify the ECG signal segments has been very expensive in time and resources. The tests that were performed can be seen in Table 2, these are not all of the experiments performed but these are the primary ones that summarize the experiments performed. From what was learned from the RF classifier the preprocessing was required to be complete before the development of the model itself, so there are no results before preprocessing to analyse. The first 3 tests with neural networks were basic Feed Forward models[3], these models gave a higher accuracy than the other tests. In spite of that, they were not useful towards the final goal as they had tendency to only select one or two classes. Therefore, another approach will be needed to build an efficient model and extra training. The main choices that were discovered were;

- Long Short Term Memory (LSTM)[3] - This model relies on memory cells that can remember an X amount of epochs into the past. Meaning as you train the model, LSTM will look into the past epochs and generate a weighting from the inherited neurons and its own formula. Therefore, this reduces the change of the missing gradient effect occurring within the layers. This model is also similar to a Recurrent Neural Network.[3]
- Convolutional Neural Network (CNN)[4] - This model is mainly used to extract features from images and process the data using pooling techniques to reduce the number of parameters.

The model that was decided to be used for experiments was the LSTM due to it being able to remember previous epochs and learn from them. This decision improved the overall model architecture as it was classifying the signals, in spite of that the accuracy was lower than first model. This shows improvement, this can be seen in test 4. However, the next stage was using layers to help improve overall model. The layers used were;

- Bidirectional[5] - This duplicates the recurrent layer and the two layers work together. This is done as the first layer generated takes in the input and as well gives the second layer generated a reverse copy of the input. This can therefore be useful for classifications sequences.
- Dropout - This was explained in the chapters Classifiers - Neural Networks. This method will help reduce bias values within the neurons causing overfitting and underfitting.

These layers were added and taken out to try and learn which would best classify the ECG segments, however, the score stays around 0.50-0.55 in accuracy. These layers were not making much of a impact from test 5. However, an unrecorded result showed a basic feed forward LSTM with only the input and output layer had a accuracy value of 0.31, this shows that the layers do improve the model, as shown in test 5, but altering the layers was not making much difference. Meaning more research into the effects the layers have will be required.

A preprocessing technique that was made improved the model but there was no record of the results. This technique was to apply a standard scalar algorithm that scaled the data to be 0 if near the mean or 1 if near the standard deviation[6]. This resulted in improving the model as a whole. Hence clarifying that preprocessing the data to support the ANN model is required to build a suitable model as if the data is not in suitable format, then to create an efficient model will become even more difficult.

The reason behind there being unrecorded test is due to sheer number of them and the tests shown in Table 2 represent a summary of the tests together and the most efficient tests of that architecture design. The experiments that were performed demonstrate the difficulty in training an efficient ANN. As these experiments took a great amount of time and research into how the individual parts of a model worked and how to improve them. Some items to improve the efficiency of the model were[7];

- Increase the size of the data
- Transform your data
- Feature Selection
- Alter the learning rate as epoch size and batch size is modified- seen in first 3 tests
- Changing batch size and epochs size
- Callbacks to reduce the chance of overfitting
- Dropout to reduce the chance of overfitting and underfitting

These are a sample of things you can do to improve a ANN[7]. During the experiments to try and build an efficient model unexpectedly the main issue that occurred was underfitting. Meaning that the accuracy of the training model was a lot higher than accuracy of the test data. This is still a problem with the final model but to a lesser effect due to Dropout layers added to the model. Therefore increasing the overall efficiency of the model and its training accuracy being closer to the test accuracy.

The score that was produced within the Kaggle competition was 0.52250 for test 9. The test was based off test 5. This score therefore shows that RF is more superior to this ANN architecture. This model uses 3 fold validation to assess the result, which come to [0.546,0.552,0.550] showing that the model is not overfitting but is bias. This results show this by main ECG Type being chosen is 'N'.

ID	Accuracy	Layers
1	0.59	Dense(50, kernel_intializer='he_uniform') Dense(4, activation='softmax')
2	0.60	Dense(100,activation='relu', kernel_initializer='uniform')) Dense(50, activation='relu')) Dense(4, activation='softmax')
3	0.61	Dense(20, activation='relu', kernel$_{i}nitializer =' random_{u}niform'$)) Dense(10, activation='relu') Dense(4, activation='softmax')
4	0.55	Bidirectional(LSTM(50, activation='relu')) Dropout(0.5) Dense(4, activation='softmax')
5	0.55	Bidirectional(LSTM(50, , return$_{s}equences = True, activation =' relu'$)) Dropout(0.5) LSTM(50, return$_{s}equences = False, activation =' relu'$) Dropout(0.5) Dense(4, activation='softmax')
6	0.51	Bidirectional(LSTM(10, activation='relu')) Dropout(0.5) Dense(4, activation='softmax')
7	0.55	Bidirectional(LSTM(20, , return$_{s}equences = True, activation =' relu'$)) Dropout(0.5) LSTM(20, return$_{s}equences = False, activation =' relu'$) Dropout(0.5) Dense(4, activation='softmax')
8	0.52	Bidirectional(LSTM(20, return$_{s}equences = True, activation =' relu'$) Dropout(0.5) Bidirectional(LSTM(20, return$_{s}equences = False, activation =' relu'$)) Dropout(0.5) Dense(4, activation='softmax')
9	0.55	Bidirectional(LSTM(50, , return$_{s}equences = True, activation =' relu'$)) Dropout(0.5) LSTM(50, return$_{s}equences = False, activation =' relu'$) Dropout(0.5) Dense(4, activation='softmax')

ID	Epochs	Batch Size	Callbacks	Activation	Notes
1	100	10	None	SGD(lr=0.01, momentum=0.9)	Only one class classified
2	60	10	None	SGD(lr=0.01, momentum=0.9)	˜ and A not classified
3	50	100	None	SGD(lr=0.008, momentum=0.9)	Only one class classified
4	200	20	None	adam	˜ not classified
5	200	20	None	adam	A not classified
6	500	20	None	adam	˜ not classified
7	500	20	checkpointer	adam	˜ not classified
8	500	20	checkpointer	adam	˜ and A not classified
9	500	20	checkpointer	adam	˜ not classified, based off test 5, uses cross validation

Table 2: Artificial Neural Network Results

4.3 Comparing Classifiers

Overall, the Artificial Neural Networks (ANN) did not compare to the Random Forest (RF) classifier. Due to the RF having the highest accuracy score of 86% and ANN best score being 55%.

In spite of that, ANN could still prove to be better than RF but this purely depends if an efficient model is generated from the alterations in the architecture inside the ANN. An architecture model that would be worth looking at would be the CNN model or a LSTM-CNN hybrid. This is due to what was found from the results of LSTM and that it may not be the most effective model in ANN to use to classify the ECG signal segments. CNN could improve the model, due to the fact that you could reconfigure the data to represent an image/graph for the CNN model to calculate the individual features within the model and classify the model through this method. Be that as it may, this approach was not looked into due time limitations but could be researched further into at a later data.

Without regard to the further research for the ANN, the current most useful model would be the RF classifier. This is due to it having a superior accuracy score. On top of this, the classifier could be easily improved due simplicity and few hyperparameters. Whereas, due to the ANN complexity it will take a greater amount of time to build a efficient model that could be equal to the current RF model or greater. This concludes that the RF would be the most efficient classifier to be used for ECG segment type selection.

References

[1] Random Forest; Will Koehrsen ; Medium Cooperation; 22/05/2019;
https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

[2] Which Machine Learning Algorithms; Paul Blondel ; SAP Conversational AI; 22/05/2019;
https://cai.tools.sap/blog/machine-learning-algorithms/

[3] Complete chart of nueral networks; Andrew Tch; Towards Data Science; 23/05/2019;
https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

[4] Convolutional Neural Network (CNN); Prabhu; Medium Corporation; 23/05/2019;
https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[5] Bidirection LSTM; Jason Browniee; Machine Learning Mastery; 23/05/2019;
https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[6] Standard Scaler; Sklearn ; 23/05/2019;
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[7] How to improve deep learning performance; Jason Browniee; Machine Learning Mastery; 23/05/2019;
https://machinelearningmastery.com/improve-deep-learning-performance/

[8] Random Forest Classifier; Sklearn; 23/05/2019;
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[9] SelectKBest ; Sklearn ; 24/05/2019;
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

[10] RandomizedSearchCV ; Sklearn ; 24/05/2019;
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

5 Appendices

5.1 Decision Tree Code

Python Version : 3.7.3 , Keras Version : 2.2.4 , Sklearn Version : 0.21.0 , TensorFlow Version: 1.13.1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
 Created on Tue May 14 18:19:26 2019

 @author: mah60
"""
import pandas as pd
import numpy as np
import csv

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RandomizedSearchCV
from sklearn.impute import SimpleImputer
from sklearn import feature_selection as fs
from sklearn import metrics

# import training and test data
train_feat = pd.read_csv("./data/ecg/train_feat.csv")
test_feat = pd.read_csv("./data/ecg/test_feat.csv")

# determine the class
y = train_feat['Type']

# determine the individual features
features = [col for col in train_feat if col.startswith('F')]

# remove misclassified features
miss_class_feat = ['F158', 'F170', 'F171','F98','F100','F144','F181','F82','F184',
                   'F4','F3']
for f in miss_class_feat:
    features.remove(f)
X = train_feat[features]

# cleans missing values within the data
imp_freq = SimpleImputer(missing_values=np.NaN, strategy='most_frequent')
imp_freq.fit(X, y)
X = imp_freq.transform(X)

# apply feature selection techniques
selector = fs.SelectKBest(fs.f_classif, k=100)
selector.fit_transform(X,y)
col = selector.get_support(indices=True)
new_feat = []
for i in col:
    new_feat.append(features[i])
```

```
X = train_feat[new_feat]
# as new features clean null values
imp_freq = SimpleImputer(missing_values=np.NaN, strategy='most_frequent')
imp_freq.fit(X, y)
X = imp_freq.transform(X)

# check if both y and X are same size
if X.shape[0] != y.shape[0]:
    raise Exception("Sample counts do not align! Try again!")
# create classifier
# ---------------------------------------------------
# find optimal hyperparameters
# create random grid for setting -
random_grid = {
        'n_estimators': [10,50,100,150,200,250,300,350,400],
        'max_features': ['auto','sqrt'],
        'max_depth': [5,10,30,50,70, 80, 90],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1 ,2 ,4],
        'bootstrap': [True, False]
                }
# classifier , with optimal parameters from RandomizedSearchCV
clf = RandomForestClassifier(bootstrap=False, class_weight=None, criterion='gini',
                        max_depth=50, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=300,
                        n_jobs=None, oob_score=False, random_state=0, verbose=0,
                        warm_start=False)
# RandomizedSearchCV used for grid search method and cross validation of each model
#rf_random_para = RandomizedSearchCV(estimator = clf, param_distributions=random_grid,
#                                    n_iter = 100, cv = 5, verbose=2,
#                                    random_state=42, n_jobs = -1)
# ---------------------------------------------------

# apply cross validation - to measure the value of the model
cross_validation = cross_validate(clf, X, y, cv=5) # gets samples from all the data
print("The 5-Fold Cross Validation Results are: {}".format(cross_validation['test_score']))

# make 80/20 split - to generate testing and training data. From known values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, train_size = 0.

# fit the data
clf.fit(X_train, y_train)
#print('----------------------------------------')
#print(rf_random_para.best_params_)
#print('----------------------------------------')
#print(rf_random_para.best_estimator_)
#print('----------------------------------------')
#print(rf_random_para.best_score_)
#print('----------------------------------------')

# make prediction
```

```
y_pred = clf.predict(X_test)

# compare results
print(classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))

# get test data and apply preprocessing
X = test_feat[new_feat]

# remove null values, with SimpleImputer
X = imp_freq.transform(X)

# predict y using the classifier
y_pred = clf.predict(X)

# collect results
output = [['ID','Predicted']]

# make 2D array to transfer to csv file
for test_id, predict in zip(test_feat['ID'], y_pred):
    #print(test_id)
    output.append([test_id, predict])

# write csv file
with open('assignment_RF_results.csv', 'w') as writeFile:
    write = csv.writer(writeFile)
    write.writerows(output)

writeFile.close()
```

5.2 Artficial Nueral Network Code

Python Version : 3.7.3 , Keras Version : 2.2.4 , Sklearn Version : 0.21.0 , TensorFlow Version: 1.13.1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue May 21 00:07:52 2019

@author: mah60
"""
import pandas as pd
import numpy as np
import csv
# data preprocessing
from sklearn.preprocessing import StandardScaler
from keras.utils import to_categorical
from sklearn.impute import SimpleImputer
# model developement
from keras.optimizers import SGD
from keras.models import Sequential
from keras import layers as l
from keras.wrappers.scikit_learn import KerasClassifier
# fiting data
```

```
from keras import callbacks as cb
from sklearn.model_selection import train_test_split
# evaluate model
from sklearn.metrics import classification_report
from sklearn import metrics
# cross validation
from sklearn.model_selection import StratifiedKFold


def load_data():
    """
    Will load the data from the test and train signal csv files.

    args => None
    Return =>
        X - np.ndarray - array of training data with selected values
        y - np.ndarray - array of expected values from the training data, transformed to
              integers depending on their key values
        X2_test - np.ndarray - test data with selected features
        reverse_class_id - dict - Y IDs but reversed
                for converting y back to orginal values
        test_ID - np.ndarray - list of test IDs for the testing data
    """
    # get test and training data
    train_signal = pd.read_csv("./data/ecg/train_signal.csv")
    test_signal = pd.read_csv("./data/ecg/test_signal.csv")

    # determine the class - the output values
    y = train_signal['Type']
    # translate classes to numbers
    # create dict
    class_id = {'N': 0, 'O': 1, 'A':2, '~':3}
    reverse_class_id = {0:'N', 1:'O', 2:'A', 3:'~'}
    # replace values
    y = y.replace(class_id)
    # determine the values
    points = [col for col in train_signal if col.startswith('X')]
    # pick out input values
    X = train_signal[points]
    X2_test = test_signal[points]

    # cleans missing values within the data
    imp_mean = SimpleImputer(missing_values=np.NaN, strategy='mean')
    imp_mean.fit(X, y)
    X = imp_mean.transform(X)
    X2_test = imp_mean.transform(X2_test)
    X = pd.DataFrame(X) # transform back to dataframe
    X2_test = pd.DataFrame(X2_test)

    # scale features to more suitable format
    std = StandardScaler()
    X = std.fit_transform(X[X.columns])
    X2_test = std.fit_transform(X2_test[X2_test.columns])
    # reshape the features to 3 dimensions [batch size, timesteps, input dim]
    X = np.reshape(X, (X.shape[0],1, X.shape[1]))
    X2_test = np.reshape(X2_test, (X2_test.shape[0],1, X2_test.shape[1]))

    # make array to create predict results labels/IDs
    test_ID = test_signal['ID']

    return X, y, X2_test, reverse_class_id, test_ID

# load data
X, y, X2_test, reverse_class_id, test_ID = load_data()

# creating call back
data_path  = 'data/ecg/lstm'
checkpointer = cb.ModelCheckpoint(filepath=data_path + '/model_{epoch:02d}', verbose=0)

# cross_valadate the model
print('-----------starting cross validation-------------')
# manually cross validation

"""
Code used to help develop the manual K fold validation ;
--------------------------------------------------------------------------------
Evaluate the Performance Of Deep Learning Models in Keras;
Jason Browniee; 23/05/2019;

https://machinelearningmastery.com/evaluate-performance-
deep-learning-models-keras/?fbclid=IwAR1ki4pVIz8qngDQ24N1ckGqAv6b
-CeWRqwo0i44ryjYsl13A6ucReSq5dQ
--------------------------------------------------------------------------------
"""
# create the splitter for 3 fold validation
Kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=7)
cv_scores = [] # used to record scores
# preset parameters
epochs = 500
batch_size = 20
test_num = 1 # record the test number
Kfold_y = y # copy y - to not alter the orginal values
is_categorical_y = False
# start cross validation
for train, test in Kfold.split(X,Kfold_y):
    # create model
    cv_model = Sequential()

    cv_model.add(l.Bidirectional(l.LSTM(20, input_shape=(1, 6000)
                    , return_sequences=True, activation='relu')))
    cv_model.add(l.Dropout(0.5))
    cv_model.add(l.LSTM(20, return_sequences=False, activation='relu'))
    cv_model.add(l.Dropout(0.5))
    #model.add(l.Flatten())
    cv_model.add(l.Dense(4, activation='softmax'))
    #opt = SGD(lr=0.01, momentum=0.9)
    cv_model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    # print k fold test number and increase for next run
    print("K-Fold Test {}".format(test_num))
    test_num = test_num + 1
    # change y to catagorical, have to do after the split
    # only do once to ensure not doing multiple to catagorical
    if not is_categorical_y:
        Kfold_y = to_categorical(Kfold_y)
        is_categorical_y = True # ensures only occurs once
    # fir the cross validation model
    cv_model.fit(X[train], Kfold_y[train], epochs=epochs, batch_size=batch_size, callbacks=
    # evaluate the score
    score = cv_model.evaluate(X[test], Kfold_y[test], verbose=0)
    # add to cv scores
    cv_scores.append(score[1]*100)
# print the overall scores as percentages
print("Scores : ")
print(cv_scores)
print('------------ending cross validation--------------')

# convert y to catagorical
y = to_categorical(y)
# create same model as the one in the cross validation section
model = Sequential()

model.add(l.Bidirectional(l.LSTM(20, input_shape=(1, 6000)
                , return_sequences=True, activation='relu')))
model.add(l.Dropout(0.5))
model.add(l.LSTM(20, return_sequences=False, activation='relu'))
model.add(l.Dropout(0.5))
#model.add(l.Flatten())
model.add(l.Dense(4, activation='softmax'))
#opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])

# make 80/20 split - to generate testing and training data. From known values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, train_size = 0.


# fit the model
model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, callbacks=[checkpointer]

# evaluate model
# make prediction
y_pred = model.predict(X_test)
y_pred = [np.argmax(pred) for pred in y_pred]
y_test = [np.argmax(test) for test in y_test]

# compare results
print(classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))

# make prediction for testing signal data
```

```
y_pred = model.predict(X2_test)
# reverse keys back to NAO~
y_pred = [np.argmax(pred) for pred in y_pred]
#y_pred = y_pred.replace(reverse_class_id)
for i, y in enumerate(y_pred):
    y_pred[i] = reverse_class_id[y]

# collect results

output = [['ID','Predicted']]

# convert to 2D array to be sent to csv file
for i in range(0, len(y_pred)):
    output.append([test_ID[i], y_pred[i]])

# write csv file
with open('assignment_LSTM_results.csv', 'w') as writeFile:
    write = csv.writer(writeFile)
    write.writerows(output)

writeFile.close()
```