Machine Learning CBC
Assignment 3: Neural Network and T-test

Group 4
Ben Shuttleworth (bms112@imperial.ac.uk)
Shuang Xia (sx813@imperial.ac.uk)
Wenke Yang(wy1713@imperial.ac.uk)
Joey Chan(ync12@imperial.ac.uk)

1 Introduction

The objective of this assignment is to study the implementation and principle behind a neural network. This is also a clarification stating that the following analysis and evaluations are all of our own work without any explicit help from other parties.

2 Part A: Artificial Neural Network

2.1 Implementation

2.1.1 Loading Data

Nothing specific about this section as we managed to load the data using the ANNdata function which is provided in the skeleton folder.

2.1.2 Creating Neural Network

Following the instruction in Part III and IV, and the MATLAB documentation, we implemented a function trainANN which can be used to train a neural network. The arguments of the trainANN function includes the examples, targets, training function, topology, parameters correspond to the training functions.

Division of data One thing worth mentioning is the train-validation-test set ratio in the testANN function. Instead of using the default divide function 'dividerand', we decided to use 'divideblock' and set the train, validation, test ratio to be 9, 1, 0 respectively. This is because in both part VI and VII, a test set is not needed in the nntool. Even in part VII, a test set is required, we performed the test outside of the network training.

Since the 9:1:0 ratio is set within the trainANN function, user does not input train data set and validation data set separately. Instead, user input the train data set followed by the validation data set as one single matrix, the trainANN

function will automatically use the last 10 percents of the data as validation set. This is how 'divideblock' handles the division of data.

For part VII, one may argue that since one fold is always for testing, and the remaining 9 folds should be divided into 8:1 manner(or anything combination which adds up to 9). However we do not see it as a problem as it is purely implementation choice, as long as we consistently use this configuration throughout the whole performance estimation.

2.1.3 Testing Network and Classification

We implemented a 'testANN' function, which inputs the trained neural network and the examples, returns a column vector of prediction.

The function makes use of the sim(net, x) function of nntool, which return a vector of output per example. After the output vector is return, the vector is applied to the function NNout2labels to return corresponding classification in the correct format requested in the CBC manual.

A sample is unlikely to be classified as 2 different classes even though the performance(ie. MSE) of 2 classes are the same. NNout2labels will returns the first index which contains the maximum value.

2.1.4 Cross Validation

In this section, we will discuss how data is divided in order to carry out cross validation. The details on how cross validation is applied in parameter estimation and performance estimation will be discussed in later sections.

crossvalind returns an array(call it X) of the same length as the number of samples in the data. Entries of the array are evenly distributed into randomized values ranged from 1 to 10. Data is then split into 10 disjoint subsets: the n-th sample in the data will be allocated into i-th set, where X[n] = i. The reason we randomized the data rather than using blocks as folds was to reduce ambiguity in case the data was ordered.

2.1.5 Maintaining the same test sets as last assignment

In parameter estimation and performance estimation, we were required to use the same test set in same fold as the decision tree algorithm. Hence in the last assignment, we stored the arrays (for both clean and noisy) returned by the crossvalind function. When we performed parameter estimation and performance estimation, we would include the index array as a guideline on how the function should decide which test set to use.

2.1.6 Evaluation of validation performance

Evaluation of the validation performance was used in both parameter estimation and performance estimation. Validation performances are essential to optimization of the parameters in network training. As explained before, the last 10

percents of examples passed into the network for training will be used as validation set. We evaluated the validation performance by setting the perfFcn in neural network to be msereg(mse with regularization). After the network training is done, a training record can be retrieved and the validation performances(ie. vperf, a column vector of mse) are included inside. In Part VI and VII, we will make use of this evaluation to decide how to find the best configuration, which will be explained later.

2.1.7 Evaluation of test performance

Evaluation of test performance is only required in performance estimation. Moreover, the performance measures we were required to report, did not specify mse, because we were more interested in the confusion matrix, precision, recall and f1 measure. Hence, although test performance can be retrieved in a similar manner as validation performance, we decided to use the testANN function to compute the predictions, and compare the predictions against the actual targets to output the performance measures required.

2.1.8 Parameter Estimation

The best configuration and best performances and best network were default empty. For each configuration we tested, we trained a network with such configuration using $trainANN$ function. we performed 10 fold cross validation to retrieve 10 validation performances in MSE. Average of the 10 MSE is calculated by dividing the summation by 10, call this the average validation performance. The best configuration and best net would be replaced by the new configuration and new net trained if the new average validation performance was better than the stored best result.
Hence, by iterating through different configurations, we can find an average test performance for every configuration. The best configuration is the configuration which yields the best average test performance(ie. the one with lowest average MSE).

2.1.9 Performance Estimation

Data was divided into 10 disjoint subsets using the method explained in Cross validation. For each fold, one set was selected to be the test set. Since test sets in different folds should be different, we decided to pick the i-th set as test set in i-th fold. Note that these sets are the one as the ones used in last assignment.

In each fold, the remaining 9 subsets would be ordered in different orders, and passed to $trainANN$, finding the best configuration for such fold using the mechanism explained in section, Parameter Estimation. Similarly, $trainANN$ would consider the last 10 percents as validation data(ie, 9 percents of the total data). Since the 9 subsets are in different order in different folds, a different validation set was used in each fold. After the best configuration was found in each fold, the corresponding best network would be used to compute the predictions for the test fold, using the $testANN$ function. And the end of the The result obtained will be further evaluated to compute different performance measures explained in the next section.

2.1.10 Compute the Average Result

We have implemented several functions to calculate the confusion matrices, recalls and precision, etc. First we calculate an accumulated confusion matrix for ten folds in total, and then we write some scripts applying functions of recall, precision and F1 based on the accumulated confusion matrix to get the average results.

2.1.11 Overfitting

We have employed early stopping and regularization in our network training process. The details will be discuss later in the report under the Question section.

2.2 Evaluation

2.2.1 Clean Data

- Confusion Matrix

	Anger Predicted	Disgust Predicted	Fear Predicted	Happiness Predicted	Sadness Predicted	Surprise Predicted
Anger Actual	91	11	6	2	21	1
Disgust Actual	16	156	4	6	14	2
Fear Actual	6	3	94	2	4	10
Happiness Actual	0	7	3	201	3	2
Sadness Actual	9	20	4	5	93	1
Surprise Actual	2	1	15	3	2	184

- Average Recall, Precision rate and F1

	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Average Recall Rate	0.6894	0.7879	0.7899	0.9306	0.7045	0.8889
Average Precision Rate	0.7339	0.7879	0.7460	0.9178	0.6788	0.9200
Average F1 Rate	0.7109	0.7879	0.7673	0.9241	0.6914	0.9042

- Average Classification Rate = 0.8157

2.2.2 Noisy Data

- Confusion Matrix

	Anger Predicted	Disgust Predicted	Fear Predicted	Happiness Predicted	Sadness Predicted	Surprise Predicted
Anger Actual	10	14	29	9	23	3
Disgust Actual	3	161	12	4	4	3
Fear Actual	3	16	129	15	6	18
Happiness Actual	2	10	12	180	2	3
Sadness Actual	4	17	25	4	50	10
Surprise Actual	2	3	17	11	6	181

- Average Recall, Precision rate and F1

	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Average Recall Rate	0.1136	0.8610	0.6898	0.8612	0.4545	0.8227
Average Precision Rate	0.4167	0.7285	0.5759	0.8072	0.5495	0.8303
Average F1 Rate	0.1786	0.7892	0.6277	0.8333	0.4975	0.8265

- Average Classification Rate = 0.7103

Evaluation of Clean Data The average classification rate is 0.8157, which is a good performance for clean data. The average recall rate, average precision rate and average F1 rate are consistent for each emotion, although they measure different characteristics. Happiness and surprise are identified best with over 0.9 on most of three measures. Followed by disgust and fear with over 0.75 on most of three measures. The identifications of anger and sad are the weakest, with roughly around 0.7 for all 3 measures.

Evaluation of Noisy Data The average classification rate is 0.7103, which is a decent rate for noisy data. General speaking, the rate is lower than that of clean data. Similar with clean data, happiness and surprise are identified the best, with higher than 0.8 in precision, recall and F1. Although most measures of both of the emotions are about 0.5 lower compared with those of clean data. Disgust is identified with acceptable rate, with even 0.07 higher in recall compared with clean set. Fear is middle-range, with about 0.6.The identification of sadness is bad, with 0.4545 average recall rate and 0.4975 average F1 rate. However, the performance of anger is really bad, with only 0.1136 average recall rate. This rate decreases the average classification rate dramatically.

Differences in Overall performance There are several differences between the performances of clean data and noisy data. Firstly, all noisy data's average rates are lower than that of clean data, about 0.1 on average. Secondly, the average recall rate, precision rate and F1 rate are very consistent for each emotion using clean data. However, the rate varies about 0.1 for each emotion using noisy data. The difference might be caused by the noisy data, which increases the possibility of identifying wrong emotions, resulting in bigger variance in recall, precision and F1 rate. Thirdly, the identification of emotion in clean data set works evenly good for each emotion while the identification quality of emotion in noisy data varies dramatically. For example, for noisy data, the best identified emotion happiness has 0.8612 average recall rate, while the worst identified emotion anger has 0.1136 average recall rate. In contrast, for clean data, even the worst identified emotion has 0.6894 average recall rate.

Difference in Anger Anger is affected by clean/noisy data set the most. It has around 0.7 recall, precision, F1 rate on clean data set, while it performs badly using noisy data, with 0.1136 average recall rate, 0.4167 average precision rate and 0.1786 average F1 rate. So anger is the easiest to be affected noise. In noisy data, it has relatively low recall and high precision, which indicates most of positive examples are correctly organised (relatively). It performs so bad partly because it has relatively few anger data. If more correct anger data exists, anger can be identified better.

Difference in Disgust Disgust is the most robust emotion. It has tiny difference between clean and noisy data in average F1 rate. We suspected it is because the a fair proportion of data is related to disgust, so it can be trained better.

Difference in Fear The difference between clean and noisy data is about 0.1, which is a normal difference. The average precision rate is even lower in noisy data, indicating emotion identified as fear is very possibly(relatively) not fear. Meanwhile, there is not as much as fear data as happiness data, so fear data is not trained as intensive as other network.

Difference in Happiness The difference between clean and noisy data is about 0.07, which is small. The impact on noisy data is small, because happiness has more than 200 samples in the data, so it has more resources for training and validation.

Difference in Sadness The difference between clean and noisy data is about 0.2, which is very big. Sadness also has a smaller example set compared to other emotions, so it is easy to explain its poor performance in noisy data. Its noisy data set might contain many noisy data to disturb the training. Its performance can be improved by providing more positive examples.

Surprise The difference between clean and noisy data is about 0.05, which is small. It is robust because it has a large example set, so it can be trained better.

Impact of Optimising Parameters On Noisy data Optimising parameter can improve the performance dramatically. We tested the program using random configurations of parameters for several times, and the average difference on performance (tr.perf) of random configuration and optimised configuration of parameters is about 5%. That indicates by optimising parameter, we can find the parameter which can provide the most accurate results.

2.3 Questions

2.3.1 Question 1

We implemented an optimization function for each of the 4 training methods. Each of them has layers of iterations to find the optimized value for each variable. Instead of optimizing the topology separately from other training parameters(ie. we did not optimize the topology with one specific training functions and claimed it works for the other training functions as well, since different training functions may have different optimized topology), we optimized the topology and training parameters together for each training function. The range of parameter to test varies, for example the best learning rate could be as small as 0.01, or as large as 3000. Since testing all the possible values would be implausible, we decided to perform a pre-learning phrase to gain some insight: We tested different parameters in a large range and attempted to narrow the best parameter in a smaller interval, hence with a narrower range, we could find a more precise solution.

Here is our general idea of optimizing parameter for traingd in pseudo code:

```
for each number of hidden neuron should be tested:
  for each learning rate should be tested:
    for each fold in the 10 fold cross validation:
      train the network with that number of hidden neurons and learning rate;
      test the performance of the trained network
      if the current performance is better than the best performance:
        best performance should be replaced by the current performance;
        best configuration should be replaced by the current configuration;
        best network should be replaced by the current network;
      end
    end
  end
end
```

We have considered whether to use accuracy, or MSE. They were chosen because of they provide a simple overview on how well a network performs, compared to recall, precision, etc, which only provides insight to a certain class. The reason we prefer MSE over accuracy is that, MSE provides information about how wrong the result is, which accuracy does not. For example, an example is classified to be class n, if the n-th entry of the output vector is the maximum. If [1, 0, 0, 0, 0, 0] is the target output, and we have 2 predictions [0.9, 1, 0.9, 0.9, 0.9, 0.9] and [0.9, 1, 0, 0, 0, 0], both of them will be classified as class 2, which they both yield the same accuracy, 0, since they both do not get the

Table 1: Best configuration found for each training method

Training Function	nH	lr	lr_inc	lr_dec	mc	delt_inc	delt_dec	MSE
traingd	15	1.4	/	/	/	/	/	0.0745
traingda	15	1.3	1	0.9	/	/	/	0.0743
traingdm	15	1.5	/	/	0.4	/	/	0.0684
trainrp	10	/	/	/	/	1	0.5	0.0703

output correctly. However using MSE, it would suggest the second prediction would be a much reasonable due to its lower MSE.

Topology Since the function is non-linear, we proposed to include hidden layers in our network training. We only experimented with 1 hidden layer because of insufficient time when we had to employ 10 fold cross validation. We also limited the range from 5 to 50 with an interval of 5. L

Gradient descent backpropagation Learning rate ranged from 0.5 to 1.5 with an interval of 0.1. We believe, learning rate is one of the parameter which its best parameter could be found in a huge range. However to save time, we have experimented with the learning rate in pre-training stage(where we tested with a larger range with larger interval to determine which interval the best learning rate lies in) and decided to keep the search range small and roughly around 1. Without further explanation, this was how we obtained the search range for others parameters

Gradient descent with adaptive learning rate backpropagation learning rate ranged from 0.5 to 1.5 with an interval of 0.1. Learning rate increase ranged from 1 to 1.2 with an interval of 0.1 and learning rate decrease ranged from 0.8 to 1 with the same interval.

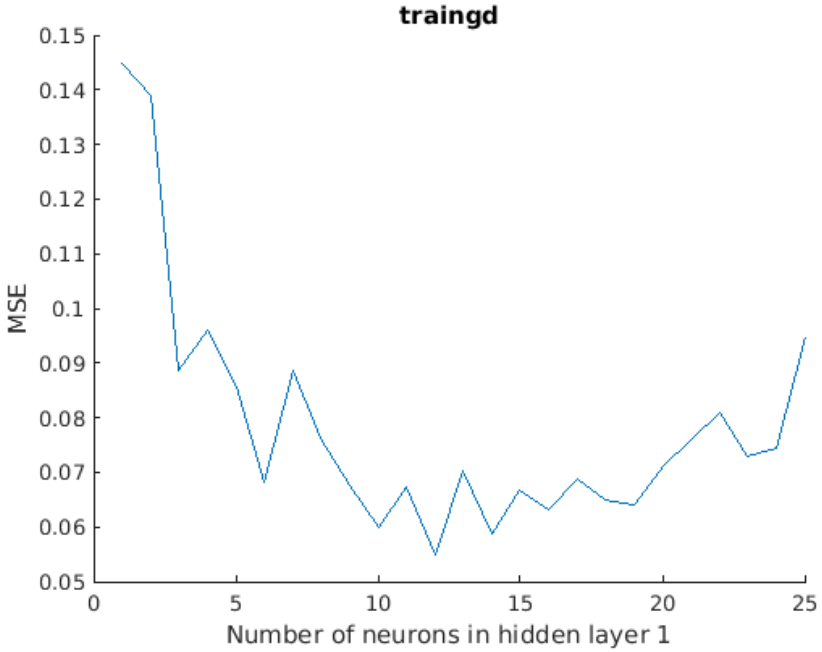
Gradient descent with momentum backpropagation Learning rate ranged from 0.5 to 1.5 with an interval of 0.1. Momentum ranged from 0.1 to 0.9 with an interval of 0.1.

Resilient backpropagation Increment to weight change ranged from 1 to 1.5 with an interval of 0.1 while decrement to weight change ranged from 0.5 to 0.9 with the same interval.

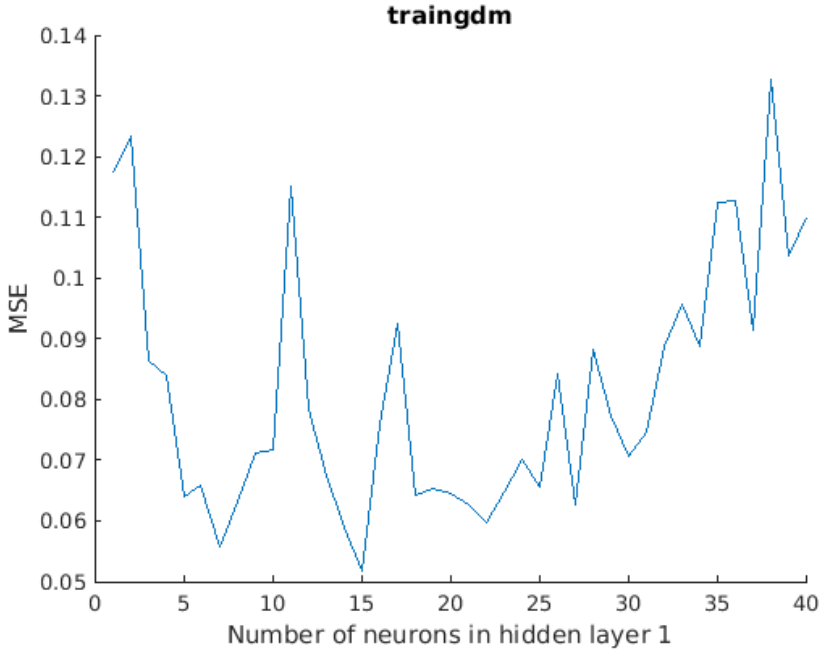
Best network Since the best network trained with trainrp has the lowest MSE(See Table 1), it will be considered the overall best network in part VI.

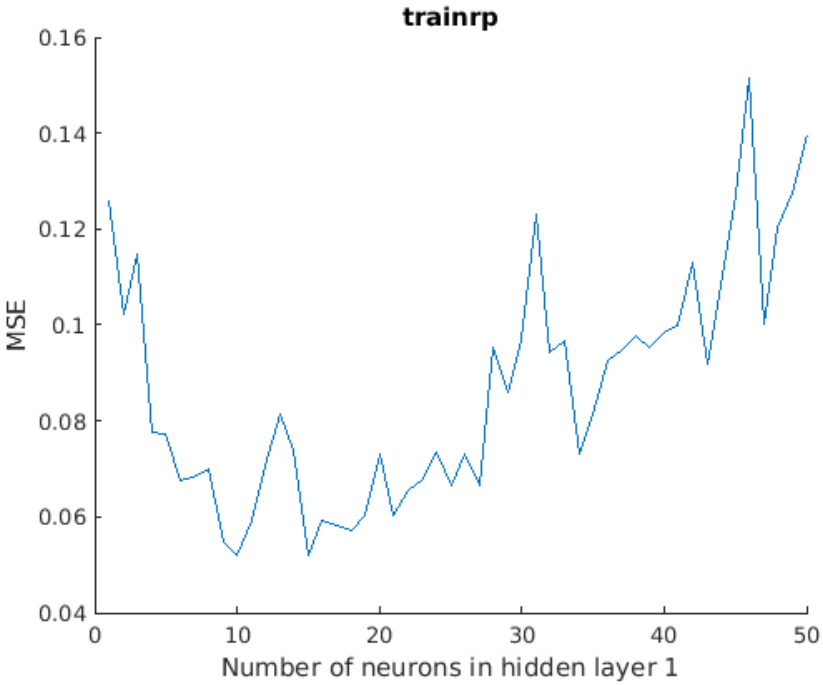
2.3.2 Question 2

To generate these graphs, we fixed all the variables using the optimized parameters excluding the topology and we experimented with the range from 1 to 50 with intervals of 1.


traingd
MSE
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
0.05
0
5
10
15
20
25
Number of neurons in hidden layer 1


traingda
MSE
0.25
0.2
0.15
0.1
0.05
0
10
20
30
40
50
Number of neurons in hidden layer 1





The performance increases (ie. average MSE decreases exponentially) until number of hidden neurons reaches around 20-30. After that, the performance decreases without any evidence of recovery. We believe the reason behind is that, if the number of hidden neurons is too small, it is likely to be undertrained, causing underfitting. If it is too large, the default early-dropout configuration would help reducing the effect of overfitting, but the decrease in performance

is still inevitable[1]. Hence we reckon a range from 1 to 50 would be an ideal range to experiment with.

As one can observed from the graph, with the best parameters we achieved in the Part VI. The best numbers of hidden neurons for traingd, traingda, traingdm, trainrp are, 12, 10, 15 and 10 respectively.

2.3.3 Question 3

Approaches employed The nntool only provides two solutions to avoid overfitting: early stopping and regularization. Early stopping was enabled by default so we did not have to explicitly do anything with it. The technique divides all data to three data sets: training, validation and test. Training computes gradient and updates weights in the network. Errors in validation set are monitored during training. The validation set error should decrease in initial training phase, but in the later stage it might increase. If the increase potential continues after specific indicated iteration, it indicates overfitting occurs, so the training should stop. In addition, overfitting is more prone to happen in multiple hidden layers network. We only has one layer hidden network, so naturally overfitting is less likely to happen.

However for regularization, we had to manually change the performFcn to msereg instead of mse. msereg is a variation of mse which adds a constant term on mse. As a result, the network has smaller weight and bias, giving smoother network responses. Other than changing the performFcn, the documentation also suggest we could train the network with other train function such as trainbr or trainbfg. However since we are limited to the 4 functions, we did not experiment with these functions.

Approaches worth considering Apart from approaches provided by nntool, we can use dropout, Max-Normal regularization and data augmentation approach. Most of these approaches aim to make weight smaller, because when weights are 0 or close to 0, it is equivalent to remove the corresponding connections between the neurons (the connections are too small to be considered.).

Dropout is an approach to change the network. During training, neurons are dropped out randomly, so there are exponential combinations of network. This prevent overfitting effectively, because there are many difference in each network's architecture and it is rare to get coincident.

Max-Normal regularization prevents overfitting by ensuring the the weights samll. It constrains the norm of incoming weights vector at each hidden unit to be upper bounded by a fixed constant c. If norm of weights is larger than c, it decrements each weight in proportion, giving final norm of incomming weights equals to c.

Data Augmentation is one of the best way to prevent overfitting. It provided more data. If there isn't more data, we can generate some data artificially (noisy data related to real world data), so that the training get used to noisy data.

2.3.4 Question 4

Scalability Using 6 single-output NNs has a higher scalability or 'mobility'. If we decided to add a new output class, we would only have to train an extra single-output network for that new class, and compare the output with other single-output networks to deduce the best prediction. However for a 6-output network, we would need to retrain the whole network, which may not be a good idea when the network is complex and takes a long time to train.

Determine output In a multiple-output network with one hot encoding, an example would be classified as class n if n-th value in the output vector is the maximum(call it m). m is basically the probability of it belonging to class n. Since a multiple network also have examples for other classes, the network does not only determine whether an example belongs to class 1 or not, it also determines if such example belongs to other classes. Thus, the probability (1-m), does not only means the probability of it not belonging to class n, it is also the probability of it belonging to another class. Semantically, it is a classification among classes, the network is answering where an example belongs.

However in a single output network, since it has no one to compare with, it does not have other examples to attempt to match it to another class(hence to improve accuracy, for example, if the network could match it to another class x, then simultaneously the network knows it is less likely to be matched to class y). The network is semantically answering a yes or no question, identifying if the example does or does not belong. The probability (1-m), solely means the probability of the example not belonging to that class. If two single output networks both decides an example could potentially belong to them, then we could not identify which class such example is more likely to belong to.

Combine the outputs of the 6 networks Since a single networks for class n would return a probability of whether the example belongs to n or not. We could simply find the class which returns the highest probability, which is similar to the approach that NNout2labels uses.

3 Part B: T-test

3.1 Part I: T-test on the clean data

Significance Level = 5%
h = 1
p = 0.0112
confidence intervals = 0.0707 and 0.4197
statistics:
- tstat: 3.1789
- df: 9
- sd: 0.2439

3.2 Part II: T-test on the noisy data

Significance Level = 5%

h = 1
p = 0.0022
confidence intervals = 0.0934 and 0.3064
statistics:
- tstat: 4.2459
- df: 9
- sd: 0.1489

3.3 Questions

3.3.1 Question 1

Which algorithm performed better when comparison was performed using the t-test(part I and Part II)? Can we claim that this algorithm is better learning algorithm than the others in general? Why? Why not?

The result of the t-test only provides evidence to support the claim that statistically, two algorithms are different. However, based on what the data suggests, since the average classification error of neural network is lower (average classification error are 6% and 8% lower in clean and noisy data) in both part I and part II. In this specific set of samples, neural network has a better performance.

However, the claim "neural network is a better learning algorithm in general" does not hold in general, especially when in part I, those data has been seen by both models. The test sets in part II would provide a better estimate on how well both specifically trained trees and network do on unseen data. However, test performance depends on the the test set. The sample size of the test set determines the reliability and accuracy of the result provided the samples are randomly drawn(eg. The result is statistically more reliable when the sample size is larger). Even though they were trained with the same set of examples, they behave differently when they see the same unseen sample. This feature enables different knowledge generalisation in different algorithms, and it should not be used as a standard to rate an algorithm.

Moreover, we need a definition on the phrase "better learning algorithm". Because an algorithm A could have a higher accuracy on a certain test set compared to another algorithm B, yet B would take less time to train. The word "better" could be in term of different scale or even a weighted combination of some. While this question does not explicitly explain the scale used to mearsure the "goodness" of an algorithm for comparison. We could only provide insights in terms of some measures.

The ultimate idea of both decision trees algorithm and neural network are both simulating and predict outputs based on acquired knowledge. However, each of them has their own advantages over the other under different circumstances. For example, A neural network could be better or worse in test set accuracy, based on how it was trained: is the topology of hidden layers good enough, or it is too complex which overfits the model? etc. Hence the performance on a limited amount of test sets could not provide a solid evidence to claim one is better than the other.

There are also other measures we could look at too: the readability of the visual presentation(ie. One could easily trace through a decision tree while one could hardly trace through a neural network with the same level of effort). To

conclude, one cannot generally rate an algorithm over the others based on a certain model.

3.3.2 Question 2

Paired-test was used. Two-sample t-test is used when the samples from the two distributions are independent(ie. the two sets of samples have no intersection with each other, say the two sets are disjoint). Paired-test is used when two sets of samples are dependent(ie. two sets of samples have intersection, say they are not disjoint).

In this context, we are using the same set of samples for both distributions with the only thing changing is the algorithm, which is every sample in the first distribution, can be paired to a sample from the second distribution, since they are the same. Paired-test would be a more appropriate choice.

However, in order to make sure they can be paired we have to make sure the same subset is used in the same fold of both algorithms. We assured this important part in our code implementation.

3.3.3 Question 3

Why do you think t-test was performed on the classification error and not the F1 measure? What's the theoretical justification for this decision?

The classification error only measures the rate at which a class is misclassified, whereas the F1 measure takes both recall and precision into account. By using the classification error instead, we get a better understanding of the general performance of the algorithm, regardless of whether it has low recall or precision.

F1 can be useful in some case, but we are not sure if the low score is due to one or more classifier useless or class imbalance. For both clean data and noisy data, the number of examples for the six emotions are not evenly distributed, some emotion has double examples of others. The case is even worse in ten-fold cross validation, because the distribution of emotions in each fold becomes even more uneven. As a result, in some fold, the data of certain emotion is very few but it classifies very precise; the other emotions have many examples, but it does not classifies well. The recall and precision in this case is strongly affected by the majority emotions and thus misleading. F1 is calculated based on recall and precision, so it will argument the misleading furthermore.

Also, if classifier is useless, when we get a high classification error, we are sure the classifier isn't useful. For low F1, we cannot know whether it is caused by classifier useless or class imbalance.

Classification error is not very robust against class imbalance and classifier useless neither. However, as it count the percentage of correct classified data over all data, it is more general and smoother, so it performs better than F1.

3.3.4 Question 4

Background Since performance of each fold is represent as a data point on the corresponding distribution, the number of folds is equivalent to the number of data point(ie sample size) in one distribution. Sample size has a positive

correlation with the degree of freedom of a distribution. In a discrete distribution which the confidence level is best calculated using a binomial distribution, having a larger sample size would provide a better simulation of a discrete distribution using binomial distribution. Thus larger sample size would suggest more reliable result.

Moreover, there may exist sophisticated analysis about how variance and mean will be affected by trade-off between reliability(ie. the number of folds) and accuracy(ie. the number of samples in each fold), but these studies provides subtle implication on their correlation, which we are not very much interested here. Hence, here we would assume, the mean and variance would lie in around the same interval(since they could be likely to spread around the same region). However, we are aware of that this could affect the result.

Increase in number of folds Increase in number of folds means larger sample size, larger degree of freedom, a similar variance and mean. Combining we have a smaller standard error of difference hence a larger t-value. it means the null hypothesis requires a to be rejected with the same significance level. In order to avoid rejecting the null hypothesis, one should decreases the significance thresholds.

However it is worth noting that, with the same total number of examples, dividing them into more folds would decrease their number in each fold. This would decrease the representativeness of one data point since each fold has less number of samples, indicating less likely to accurately representing the whole population. Hence the overall impact cannot be determined.

Decrease in number of folds For similar argument, a smaller sample size would suggest a smaller degree of freedom, a similar variance and a similar mean. This would lead to increase in standard error of difference hence a smaller t-value. It is less likely to be rejected with the same significance level hence one could increase the significance threshold.

Again, increasing the number of samples in each fold would improve the accuracy of one data point, since a larger sample size in a fold would representing the whole population more accurately. Hence the overall impact once again cannot be determined.

3.3.5 Question 5

For both of algorithms, retraining is required from initial stage.

For **Decision Tree Algorithm**, all decision tree need to be retrained, because the best classifier for each emotion might change after a new emotion is added. Function choose_best_decision_attribute is used to select best classifier for each emotion, it selects the attribute with the highest information gain. Since information gain is based on the impurity of attributes in examples, and impurity depends on number of positive and negative examples for each emotion, as new

emotions added to database, the number of positive and negative examples for each motion will definitely change.

For example, attribute 1 might be a good classifier for happiness, and the first branch of happiness tree is attribute 1. In old case, as long as attribute 1 is true, we can classify that emotion as happiness. After the new emotion is added, attribute 1 classify happiness and the new emotion in the same way. We have to develop the decision tree further in order to classifies all emotions correctly.

Therefore, re-selecting the best attribute and retrain the decision tree using new best classifier is needed after including new classes.

For **Artificial Neural Network Algorithm**, when new emotion is added, this is equivalent to having an extra output in the output layer of the network. Since this is automatically catered by the neural network toolbox, we would not have to take extra consideration into this. However, a new neural network is required to be trained, so that the weight can be adjusted correctly. When the number of network output changes, in order to maintain the same level of accuracy, we would have to find a new set of best configuration to train the network. When there does not exist a general formula to find the best configuration, it could only be done by experimenting with different configurations. Apart from the parameter issue, since we had the foresight to see this problem coming, we implemented most of our functions in a general way that they could handle different number of output, so in implementation-wise, we would only need to change the variables which is related to the output.

References

[1] comp.ai.neural-nets FAQ, Part 3 of 7: Generalization Section - How many hidden units should I use?, http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html

[2] Matlab Documentation, $http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf$