Bagging:

1.1:

Bagging or Bootstrap aggregating is meta-classifier designed to improve the stability of classification tree (it was initially invented for tree classifier, but later researchers expanded this idea and now theoretically it can be used on any classifier.). It uses Bootstrap to generate m sets of data points. Then we can fit a classification tree on each of the data set. So we will have m tree models. Then we make prediction based on the average result of those m tree models. The Weka class name for bagging is weka.classifiers.meta.Bagging.

1.2:

There are three parameters to adjust: size of the bag, number of iteration, and base classifier.

Size of the bag: this parameter controls the size of the data set for each tree model. If the bag is too small, then each tree may be biased toward the population. If the bag is too big, the computation time can be very long.

Number of iteration: this parameter controls how many tree models we should have. If the number is too small, then bagging will be less stable. If the number is too large, then the computation time can be very long.

Base classifier: this parameter controls the type of the tree models e.g. J48, random tree, random forest etc. An appropriate base classifier can help bagging give more accurate result since bagging just take the average of the tree model results. The learning time and classifier size of bagging are dependent on the base classifier.

After doing CV Parameter Selection, it turns out that the default setting of size of bag and number of iteration is already optimal. The error rate and robustness are very insensitive to those two parameters when they are large enough. The default setting for size of bag and number of iteration is 100 and 10 respectively. The error rate and robustness are very sensitive to the base classifier. The default base classifier REPTree is not very good. I tried several other tree classifiers including random forest, random tree, J48 and many other trees. For the same data set, different base classifier can give very different performance, and there is no classifier that works best for all data set. So I have to use internal CV to select base classifier for each data set. Time taken to build model: 0.13 seconds in Weka.

1.3:

My final setting for the base classifier is internal CV on random forest and J48, i.e. dynamically pick the classifier which performs best in the CV of training data set. All other parameters are using default setting since CV Parameter Selection shows that they are already optimal.

Random Forest:

1.1

Random Forests are an ensemble learning method for classification (and regression). It is operated by constructing a multitude of decision trees at training time. For each tree, when splitting a node, only a randomly chosen subset of the dimensions is considered. The prediction output of the decision tree is the mode vote of all trees. The Weka class name is weka.classifiers.trees.RandomForest.

1.2

There are three parameters that can be tuned: the number of trees to build, the number of features to consider, and the maximum depth of the trees.

The **number of tress**: it controls how many trees are generated in this forest. The bigger this number is, the longer time learning and predicting procedure will take and the bigger the classifier size will be. At the same time, more tress can make the forest have less over-fitting risk.

The **number of features**: for any tree, when splitting a node, only a randomly chosen subset of the dimensions is considered. This parameter is used to control the size of dimension subset. This parameter will not affect the classifier size. The smaller it is, the less learning time is needed, the less over-fitting this classifier tends to be. However, if it is too small, classifier will be less accurate since we do not have enough features to make node split decision.

The **maximum depth of the trees**: it controls maximum depth of all trees in this forest. The bigger this number is, the bigger classifier size tends to be and the longer learning procedure tends to take. Moreover, a smaller depth tends to have less over-fitting risk, but if it is too small, classifier will be less accurate since we do not have deep enough decision path.

In our experiment, we find that when number of trees is small, Random Forests' performance is very sensitive to it. For example, when we increase the number of trees from 3 to 30, the error rate declines by 15%. However, as the number of trees getting bigger and bigger, the performance improvement starts to gradually fade away. From 75 to 100, the error rate stays nearly the same, varied only by 1 or 2 percent. The Random Forests seems not very sensitive to "number of features" parameter. It is optimized around the value log(Dimensions)+1, but deviating far from that only cause the error rate increase by 2-4%. The maximum depth of the trees is optimized when we set it as unlimited, but a relatively big enough number (e.g. 15) is also nearly equally good. When it is set too small (e.g. 3), the error rate is increased by 5 to 20% across different data set.
The training time in Weka is 0.04 second.

1.3

After experiment, we find the best option combination (giving us the lowest error rate) is "number of trees = 75", "number of features = log(Dimensions)+1", and "maximum depth of trees = unlimited".

Random Tree:

1.1

Random Tree is a special decision tree. The specialty is when generating Random Tree, a subset of attributes is randomly chosen and then used for split decision. Moreover, Random Tree performs no pruning. The Weka class name is weka.classifiers.trees.RandomTree.

1.2

There are 4 parameters to set: number of attributes, minimum number of instances, maximum depth, Number of folds for backfitting.

number of attributes: it controls the number of attributes to randomly investigate when a node is splitting. This parameter will not affect the classifier size. The smaller it is, the less learning time is needed, the less over-fitting this classifier tends to be. However, if it is too small, classifier will be less accurate since we do not have enough attributes to make node split decision

minimum number of instances: it controls the minimum number of instances per leaf. The bigger this number is, the smaller classifier size is and less learning time is needed. Moreover, higher minimum number of instances on leaves can lessen over-fitting risk, but when it is too high, many nodes may lose the chance of correctly splitting further.

maximum depth: it controls the maximum depth of this tree. The bigger this number is, the bigger classifier size is and the more learning time is needed. Also a deeper tree has more risk of over-fitting, but if it is too small, classifier will be less accurate since we do not have deep enough decision path.

Number of folds for backfitting: it controls number of hold-out folds used for estimation of class probabilities. The more folds we used for backfitting, the less learning time is needed and the smaller classifier size is. Moreover, too many data hold for backfitting may cause higher over-fitting risk.

In our experiment, the Random Tree seems sensitive to “number of attributes” parameter. It is optimized when equals to the number of all attributes. The smaller value it is, the bigger value error rate has. The maximum depth of the trees is optimized when we set it as unlimited, but a relatively big enough number (e.g. 20) is also nearly equally good. When it is set too small (e.g. 2), the error rate is increased by 5 to 20% across different data set. The classifier performance is also sensitive to “minimum number of instances”. A moderate value of it may be good to prevent overfitting. However, if the value is too big, the error rate will rise sharply. (e.g. when it is set to 20, the error rate is rose by almost 20% comparing to setting it to 1). The classifier performance is also sensitive to “Number of folds for backfitting”. When it is too big, the error rate will boost drastically. (Error rate is more than 10% higher when setting it to 10 comparing to leaving it as 0). The training time in Weka is 0 second (too small, so out of precision).

1.3

After experiment, we find the best option combination (giving us the lowest error rate) is “number of attributes = number of total attributes”, “Number of folds for backfitting = 0”, “minimum number of instances = 2” and “maximum depth = unlimited”.

Summary:

After carefully tuning the parameters in all the three classifiers, we find that bagging (on J48 and Random Forests) is the most robust classifier. By tuning, we improve the average error ratio from 0.767 to 0.666 and the max error ratio from 1.08 to 1 for bagging. In addition, the average error ratio in milestone 3 is better than the average error ratio in milestone 2. We improved from 0.691 to 0.666.