Development of Automated Methods for Using Satellite Imagery to Monitor Changes in Vegetative Communities

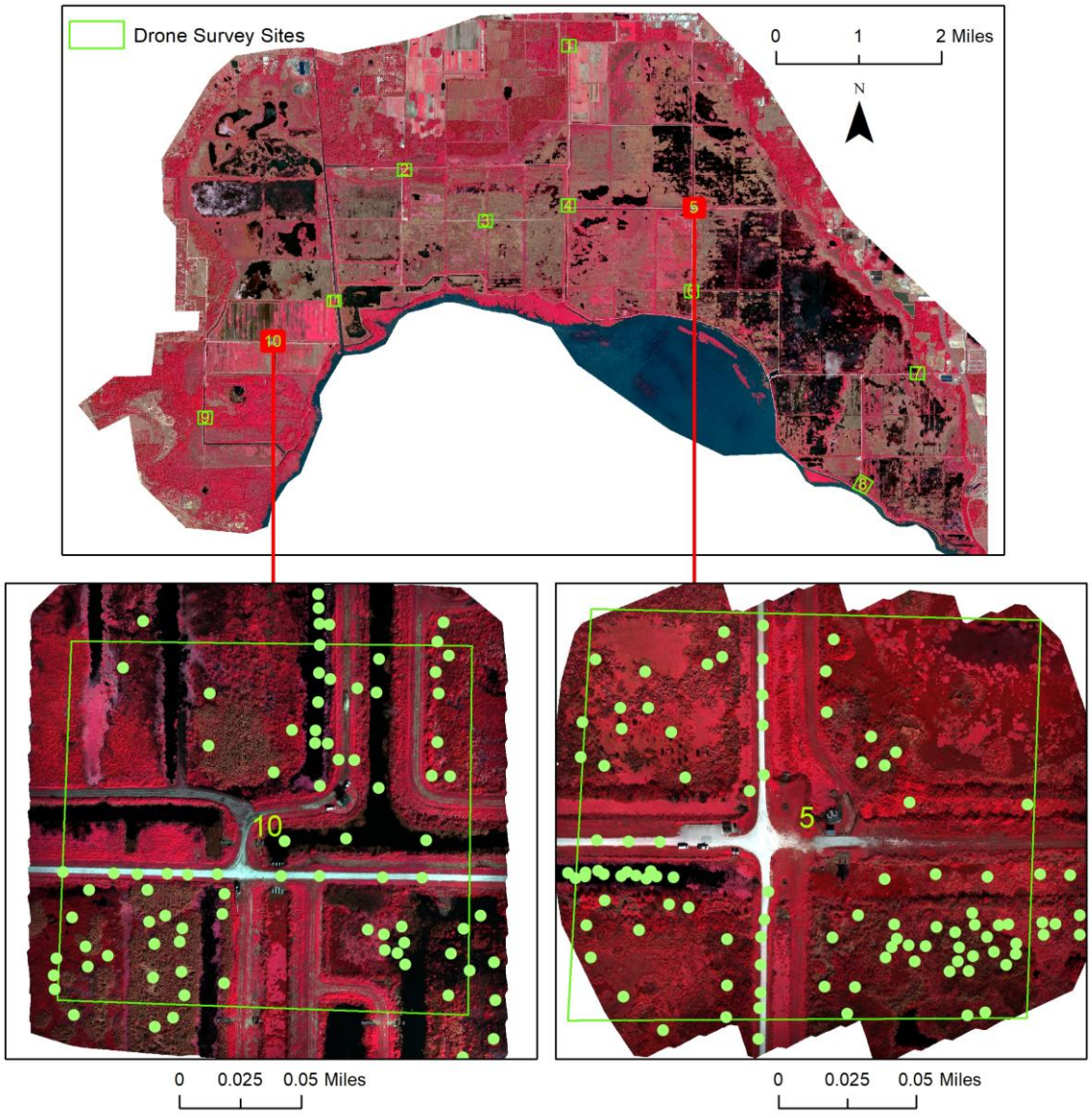
Report for Quarter 2/Task 6 in 2021-2022

Submitted to: St. Johns River Water Management District

Caiyun Zhang, PhD

Department of Geosciences, Florida Atlantic University

777 Glades Road, Boca Raton, FL 33431, USA, Tel: 561-297-2648; Email: czhang3@fau.edu



Drone survey sites at Lake Apopka North Shore (LANS) shown as a color infrared composite of WorldView-2 (top), and two sample drone imagery products (bottom). Manually collected drone reference points are in green dots.

Contents

In this quarter, we remapped Lake Apopka North Shore Area (LANS) using the 2021 WorldView-2 (WV-2) imagery, LiDAR DEM, newly collected field samples, drone samples, and WV-2 reference samples. We tested the classifiers using independent field and drone samples, i.e., trained the classifiers excluding field and drone samples, as suggested by the District in the last Quarter so that independent datasets are used to test the classification maps. We generated drone imagery products, revised the tutorial based upon comments from the District, and developed an *R* script and tutorial to run the machine learning ensemble models in *R*, rather than WEKA.

1. Unmanned Aerial Vehicle (UAV) Data

1.1 Drone data collection

We collected UAV (also commonly known as drone) data during the field visit 11/15/2021-11/18/2021 using a DJI P4M system. This system has a maximum flight time of 27 min and collects data using a RGB camera and a multispectral camera array covering Blue, Green, Red, Red Edge, and Near Infrared bands (Figure 1). We effectively collected data for 11 sites but failed for one upland site due to technical issues on the first day visit. The distribution of the 11 drone sites is displayed in Figure 2. We collected Ground Control Points (GCPs) at each site for drone data processing (Figure 3).

Figure 1. The DJI P4M drone operated by FAU students in the field (left, and right bottom), and the five multispectral cameras and 1 RGB camera onboard the DJI P4M drone (right top).

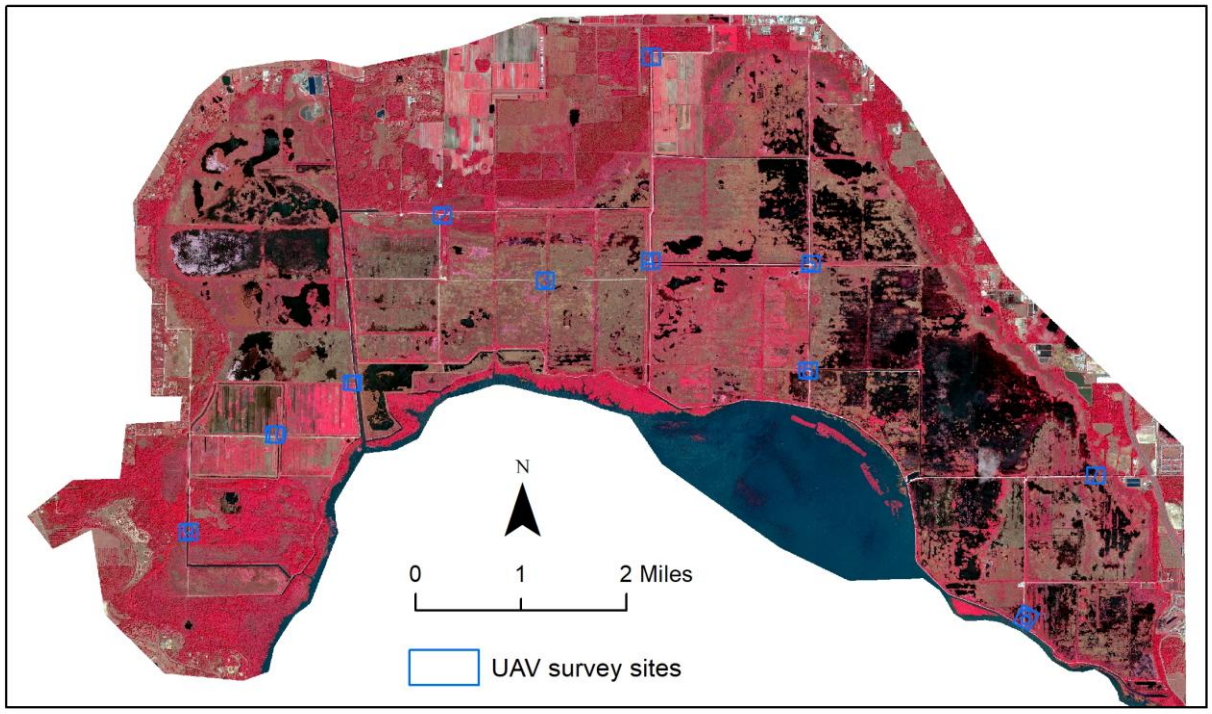


Figure 2. Drone survey sites overlying the color infrared composite of 2021 WV-2 imagery.

Figure 3. Collection of GCPs at the drone survey sites.

1.2 Drone data processing

After field drone data collection, we applied the standard UAV data processing procedure integrated in the Pix4D mapper software package to process the data and generated drone imagery products. The final drone imagery products have a spatial resolution of 0.16 m, a spectral resolution of 5 bands (Blue, Green, Red, Red Edge, and Near Infrared), and a projection of NAD 1983 UTM Zone 17 N. The maps of these 11 drone image products are shown as a natural color composite in Figures 4-6.

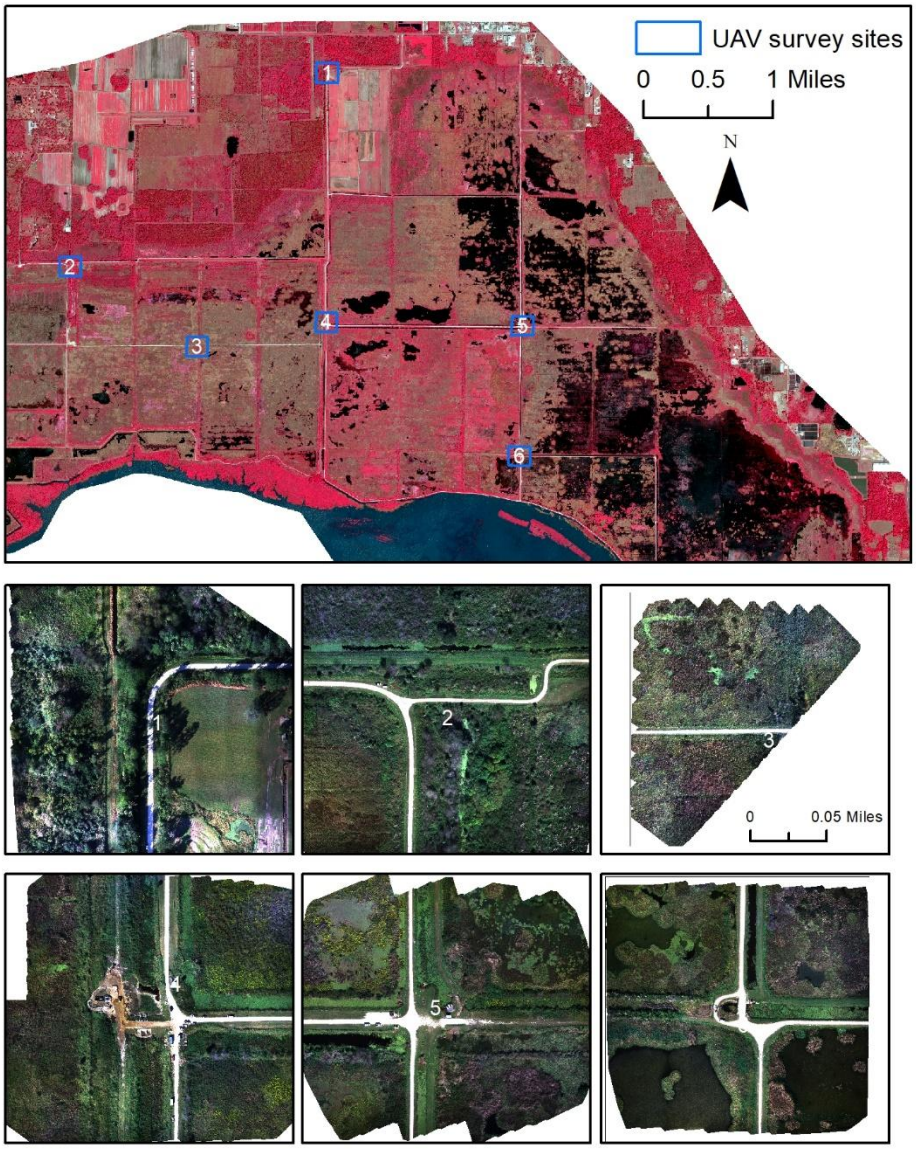


Figure 4. Drone imagery products for sites 1-6.

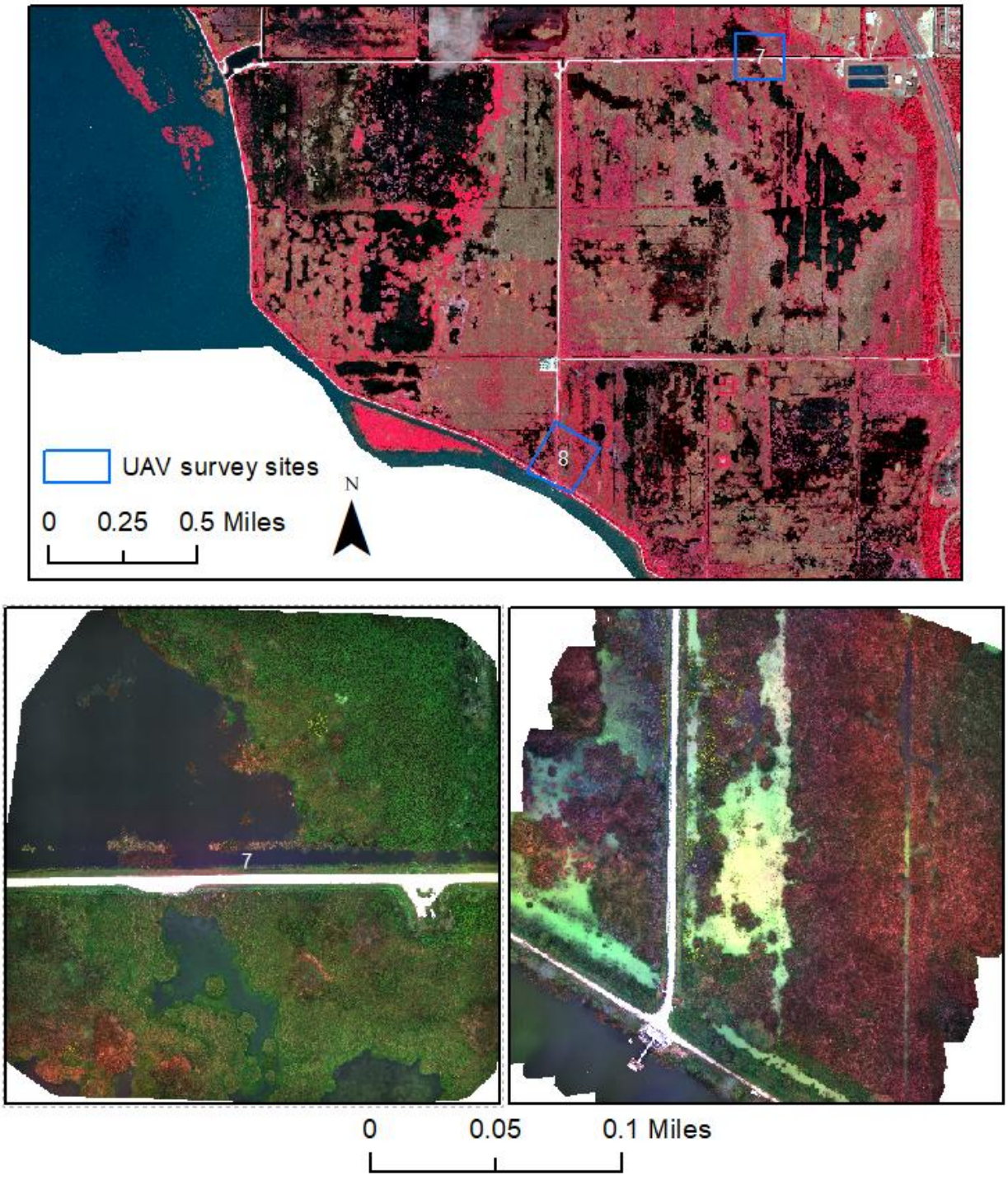


Figure 5. Drone imagery products for sites 7 and 8.

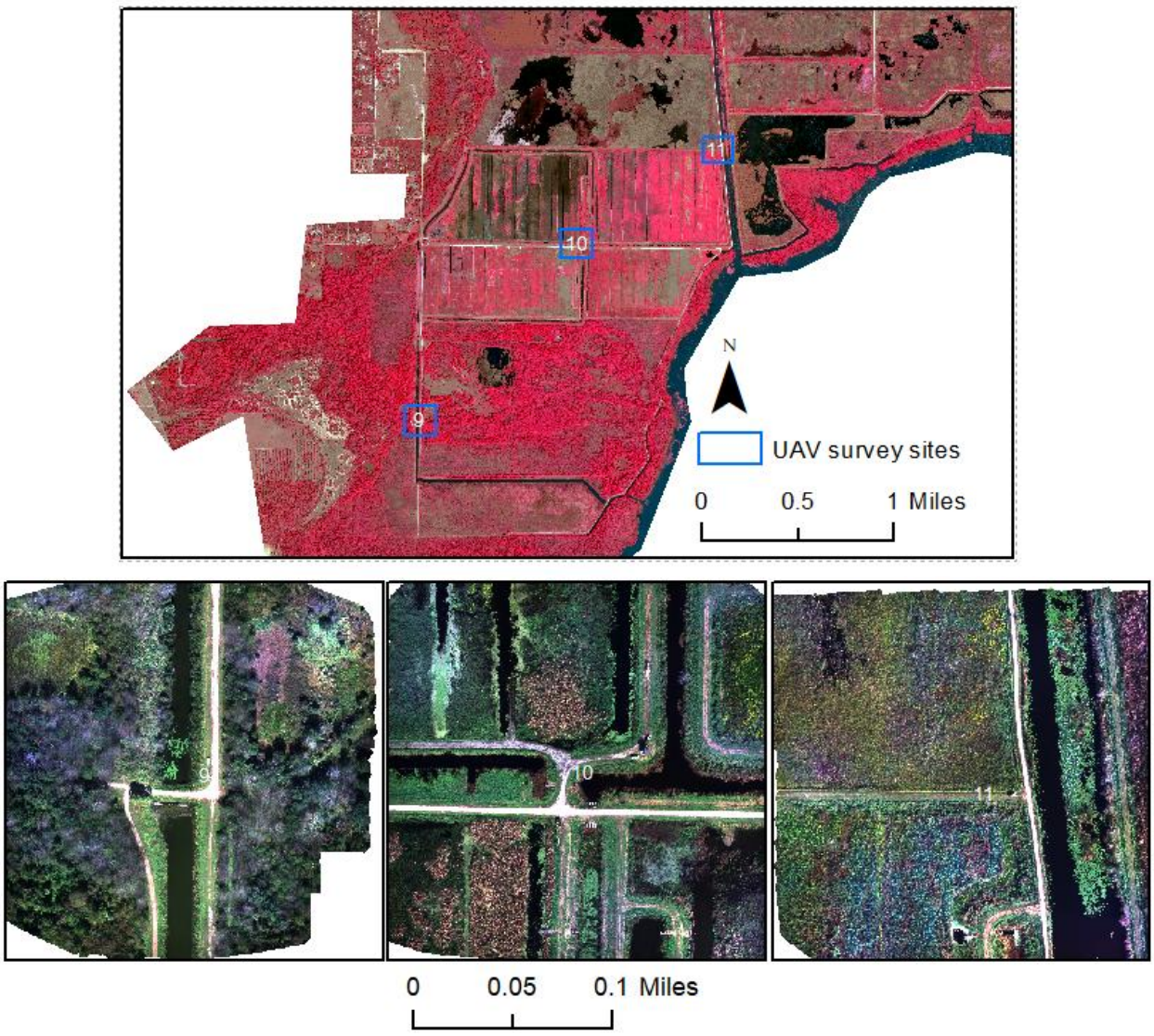


Figure 6. Drone imagery products for sites 9-11.

2. Testing using Field and Drone Samples

2.1 Testing using field samples

The field visit enhanced our capacity to interpret the fine resolution WV-2 imagery. We thus recollected WV-2 based reference point samples following the procedure reported in Quarter 1 2020-2021, but this time the field samples also guided the imagery sample interpretation. We believe the imagery-based reference samples are more accurate than the samples we used in the last Quarter. We then applied the WV-2 imagery-based reference samples as the training samples, spatially joined these samples with the image segmentation file, and identified the training objects. We conducted machine learning classification and tested the classification results using point samples collected in the field. Similarly, field collected point samples were spatially joined to the segmentation file and testing objects were identified. Note that field

collected samples were not used in the training, instead they were used as an independent dataset to test the classifiers. We mapped 18 communities and clouds appearing on the imagery, but did not map Willow Swamp (WS), Scrub (SC), and Spartina (SP) due to limited training samples. In the field we collected samples for 19 communities but did not collect samples for Dry Prairie (DP), and Pine Flatwoods (PF) which were mapped. Therefore, we could only test the mapped communities for which we had field samples. The newly collected WV-2 based reference samples are displayed in Figure 7, and the field testing samples are displayed in Figure 8.

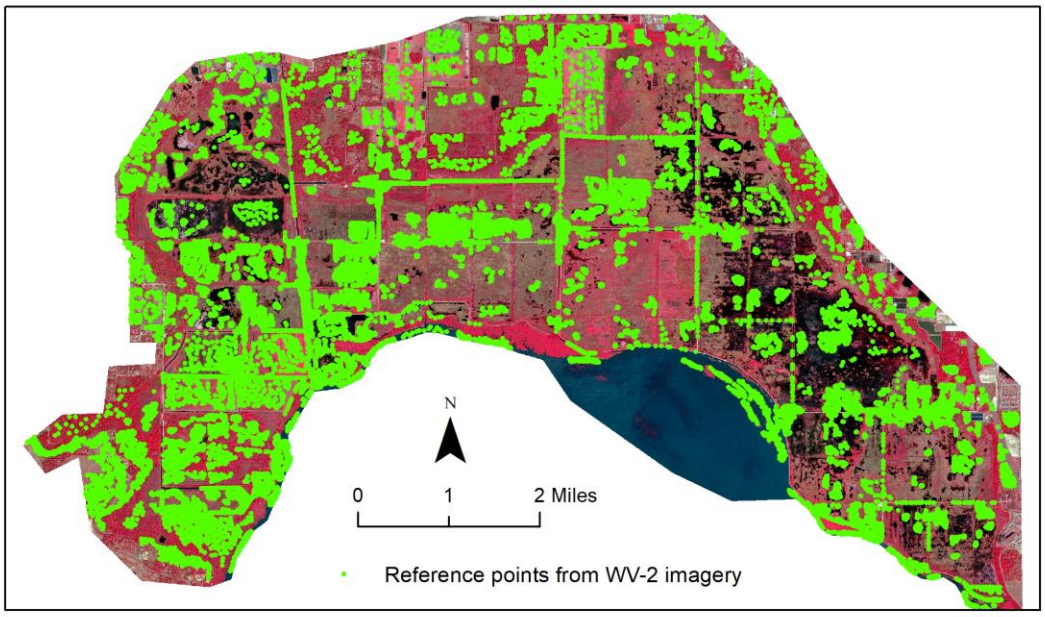


Figure 7. The newly collected WV-2 based reference samples.

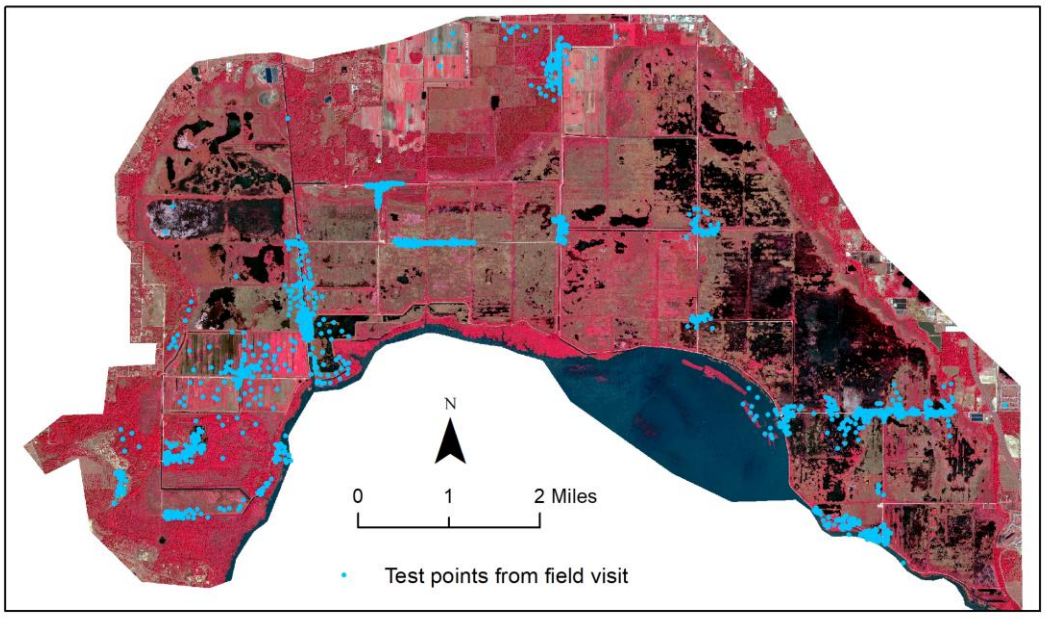


Figure 8. Field collected testing samples.

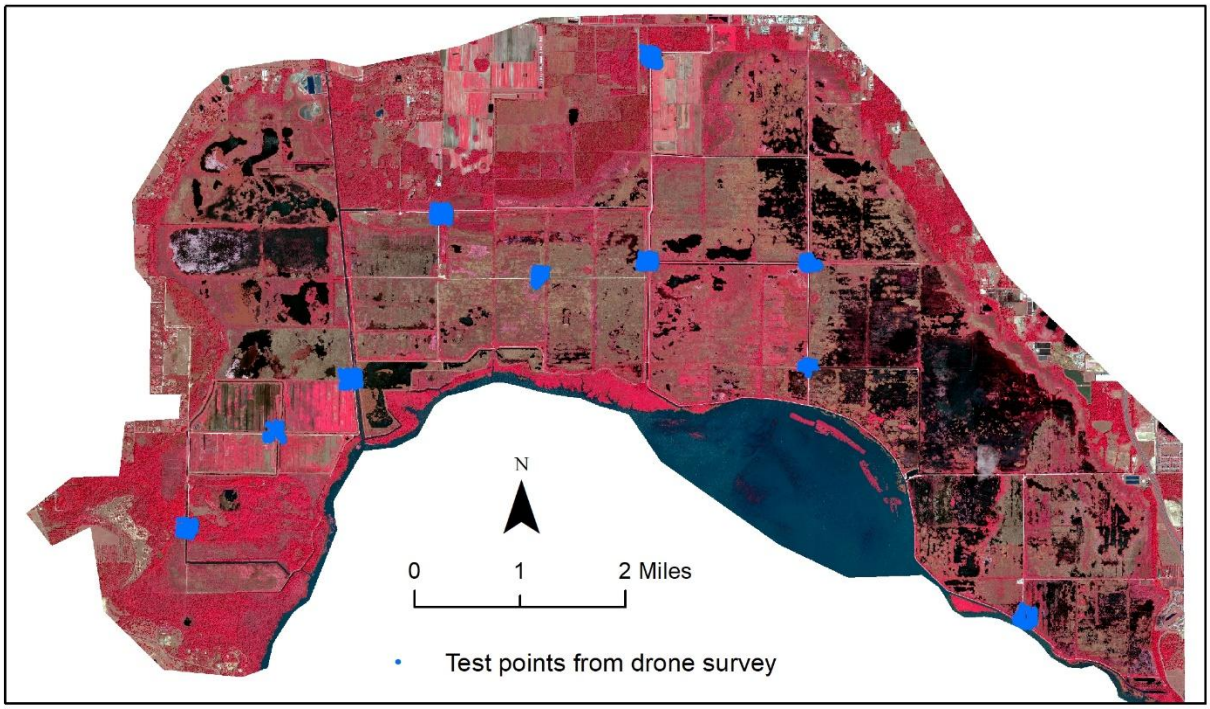


Figure 9. Collected drone reference samples.

The field testing results revealed an Overall Accuracy (OA) of 68.2% based on the ensemble analysis of Artificial Neural Network (ANN), Support Vector Machine (SVM), and Random Forest (RF) (Table 1). Further examination of the error matrix revealed that a low accuracy was obtained for Cattail (CT) which was not mapped on the reference map of 2017. Based on the crosswalk excel file sent by the District, areas with original Shallow Marsh should be further refined into Grass/Sedge (GS), Mixed Herbaceous Marsh (HM), Cattail (CT), Phragmites (PH), or Sawgrass (SG). This created a challenge to manually select GS or CT training samples on WV-2 imagery based on the original reference vegetation map, thus GS and CT training samples were mainly collected in the field. GS and CT are both a component of HM. When finer-level plant community identification is required than is present in reference maps, more extensive field work may be needed to correctly assign communities. The error matrix of the testing results using field samples is separately uploaded.

Table 1. Testing results for each machine learning classifier and ensemble analysis for LANS; EA is ensemble analysis of ANN, SVM, and RF.

Testing using field reference samples					
	ANN	k-NN	SVM	RF	EA
OA (%)	60.3	51.6	66.4	70.9	68.2
Kappa Value	0.54	0.43	0.61	0.66	0.63
Testing using drone reference samples					
OA (%)	66.6	62.6	71.6	67.9	71.3
Kappa Value	0.57	0.53	0.63	0.59	0.63

k-fold cross validation using all combined reference samples					
OA (%)	72.4	67.6	76.5	76.2	77.3
Kappa Value	0.69	0.63	0.73	0.73	0.74

2.2 Testing using drone reference samples

Similarly, we trained the classifiers using WV-2 imagery reference samples and tested the classification results using drone reference samples only. We manually collected point samples from drone imagery which helped identify more reference samples due to its finer spatial resolution. We spatially aligned the field collected samples with the drone imagery in ArcGIS, and also loaded in the 2017 aerial photography derived vegetation map, then manually interpreted point samples (Figure 9). Only 11 communities occurred at drone surveyed sites, and thus we could only test the accuracy for these communities. The testing results are listed in Table 1. SVM achieved the highest accuracy and k-NN produced the lowest accuracy. Ensemble analysis of SVM, ANN, and RF obtained an OA of 71.3% and Kappa value of 0.63. Similarly, CT was confused with HM; Hydric Hammock (HH) was confused with Shrub Swamp (SS) and (Uplands) UH, leading to lower accuracies for these communities. This is consistent with the results reported in last Quarter. To increase accuracy, either more sites should be flown, or larger areas should be covered with each flight. The error matrix of the testing results using drone reference samples is separately uploaded.

2.3 *k*-fold cross validation

We also combined all reference samples collected from WV-2 image, field, and drone images into one training sample file, and ran the *k*-fold cross validation. This could test all 18 mapped communities and cloud. The testing results are also displayed in Table 1. SVM and RF classifiers obtained a comparable OA larger than 76%, and *k*-NN produced the lowest OA again. Ensemble analysis of ANN, SVM and RF slightly increased the OA to 77.3% with a Kappa value of 0.74. Analysis of the error matrix, again, revealed the lowest accuracy for identifying GS which was confused to HM, SS, and Levee/Road (LR), like the findings in last Quarter. GS and CT were not mapped in the 2017 aerial photography derived map, and needed to be delineated from the general community called Shallow Marsh in the old name system. Collection of GS and CT reference samples was thus mainly based upon field visit. Again, GS, CT, and HM were confused, which is understandable, since Cattail and Grass/Sedge marsh can be a component of HM. HM was also classified as Shrub Swamp (SS) and small shrubs can be a component of HM communities as well. In the field we saw many patches of a mixture of HM and SS, and thus the confusion is understandable given the lack of a hard boundary between the two communities. The classification accuracy of GS, again, was very low (6.5%) in the ensemble model. GS was also classified as levee (levee with grass can be confused with GS), CT, and HM. As discussed in the last report, this is understandable given that CT is grass-like and both grasses and sedges are components of herbaceous marshes. We had very limited training samples of GS. In the field we did not see many homogeneous GS areas. The classification maps (Figure 10)

show a very small coverage of GS, therefore maybe GS can be eliminated in the classification. The error matrix of *k*-fold cross validation is separately uploaded.

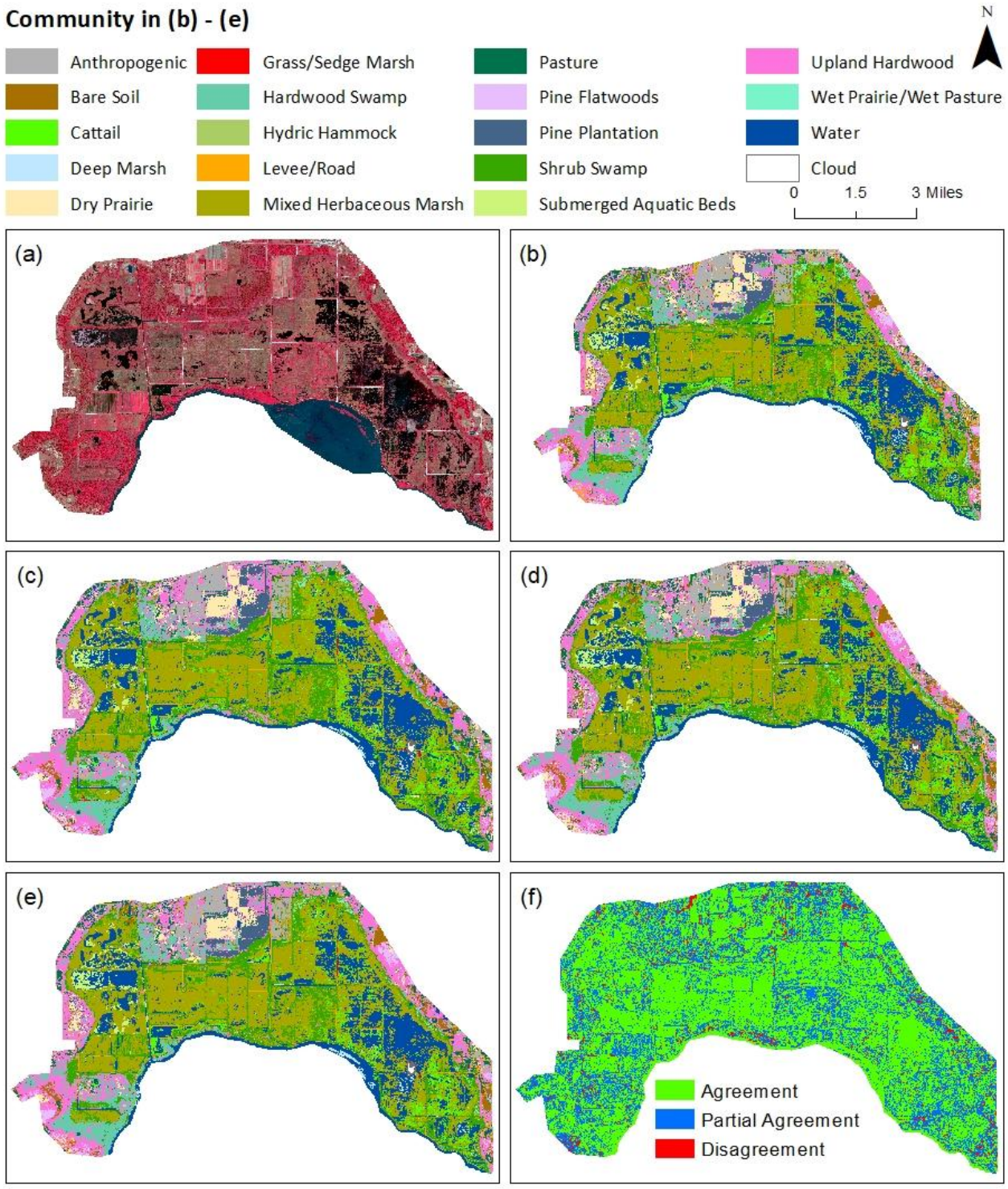


Figure 10. (a) 2021 WV-2 pansharpened imagery; Object-based classifications for 2021 imagery from (b) SVM, (c) RF, (d) ANN, and (e) Ensemble analysis. (f) Uncertainty from ANN, SVM, and RF.

3. Classification and Mapping

We combined all WV-2 image-based reference points, field and drone collected points into one file to train the classifiers and generated final classification map products, as shown in Figure 10. In general, all classifiers produced a similar spatial pattern. Pine Plantation (PI), Pine Flatwoods (PF), Cattail (CT), and GS were delineated in the 2021 classified map. These communities were not mapped in the 2017 reference map. For most regions, three classifiers (ANN, SVM and RF) generated consistent classifications, as shown in green in the uncertainty map in Figure 10 (f). Red regions show a "warning sign" where the classification from each classifier was completely inconsistent with each other and can be used as an indicator of where further training points should be gathered.

4. *R* Code and Tutorial Development

We can also use the open-source *R* package to run the machine learning training, testing, and classification. The open-source WEKA package requires a manual set of parameters needed for each machine learning algorithm, and also involves some manual steps to get the error matrix, and final classification outputs in the format we need to import into ArcGIS. In addition, WEKA cannot conduct the McNemar significance test and ensemble analysis. We developed the *R* code to automate parameter tuning of each machine learning algorithm. We also developed McNemar test codes, error matrix output, and ensemble prediction codes. The developed *R* code and tutorial is attached in **Appendix 1**, and were uploaded as separate files to the District's ftp server.

5. Files for Quarter 2, 2021-2022

Products in this quarter include ArcGIS shapefile of ANN, SVM, RF, Ensemble Analysis classification, Uncertainty Analysis; three error matrixes in Excel format; drone imagery products; developed *R* code, testing data, and tutorial. They are uploaded to the District's ftp site. Table 2. summarizes the deliverables for this Quarter.

Table 2 Deliverables in Quarter 2, 2021-2021.

Products	File Name/format	Description
Map dataset	ArcGIS shapefile: ANN.shp; SVM.shp; RF.shp, EA.shp, Uncertainty.shp	Classification results using the combined reference samples (drone, field, and WV-2 reference points) to train the classifiers
Drone imagery products	TIF files: Site_1.tif, Site_2.tif…Sites_11.tif	Drone imagery products in TIF format for sites 1-11
Error matrix	Excel files: ErrorMatrix_Field.xlsx; ErrorMatrix_Drone.xlsx ErrorMatrix_All.xlsx	Testing results using field data, drone reference data, and k-fold cross validation of all combined reference data
R code and tutorial	Text format: ML_Zhang_V5.R; MachineLearningTutorial_R.docx Training.csv; InputForPrediction.csv	Developed R code for machine learning training, testing and classification, McNemar test, and ensemble analysis, a tutorial, and two input testing files.
Report	Word file: Q2_2021-2022_Report_V1.docx	Quarterly report in MS Word format.

Appendix 1 Machine Learning Classification in R

Machine Learning Classification and Mapping using R

Caiyun Zhang

3/24/2022

This tutorial provides steps for conducting machine learning (ML) and classification in open source package R. We use the exported training data from ArcGIS in csv format to develop machine learning models (the file has attributes used to predict the class, and the class attribute for training samples) and the exported csv with unknown class target as the input data for final prediction. We use RStudio and caret package in R to conduct this critical step as an alternative to WEKA. The steps from RStudio installation to final prediction using ML models are described below.

1. Install R and RStudio on Windows 10

First we need to download and install R. Click on the following link: https://cran.r-project.org/bin/windows/base/ . Now click on the Download R for windows (Figure 1). It will take some time to download.

Figure 1 Download R for windows.

Now right click on downloaded file> run as administrator> press on the next button and keep to default> select attribute table page click on create a desktop shortcut> now it will start the installation process.

After the installation of R, now we need to install R studio. Click on the following link: https://www.rstudio.com/products/rstudio/download/

Now click on the free download button.

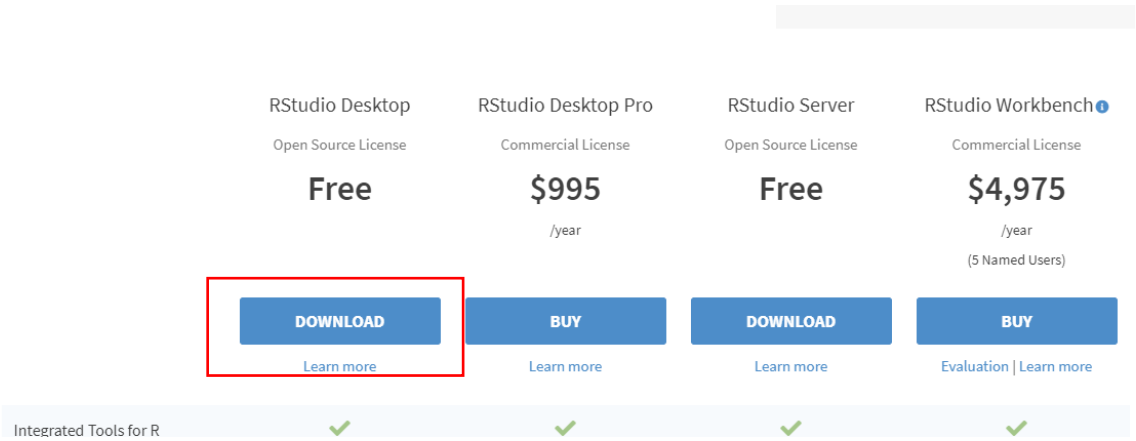


Figure 2 Download RStudio.

It will redirect you to RStudio Desktop page. If you scroll down you can see setup file of RStudio for windows, macOS, Ubuntu and etc. Scroll up again and click on Download RStudio for windows button.

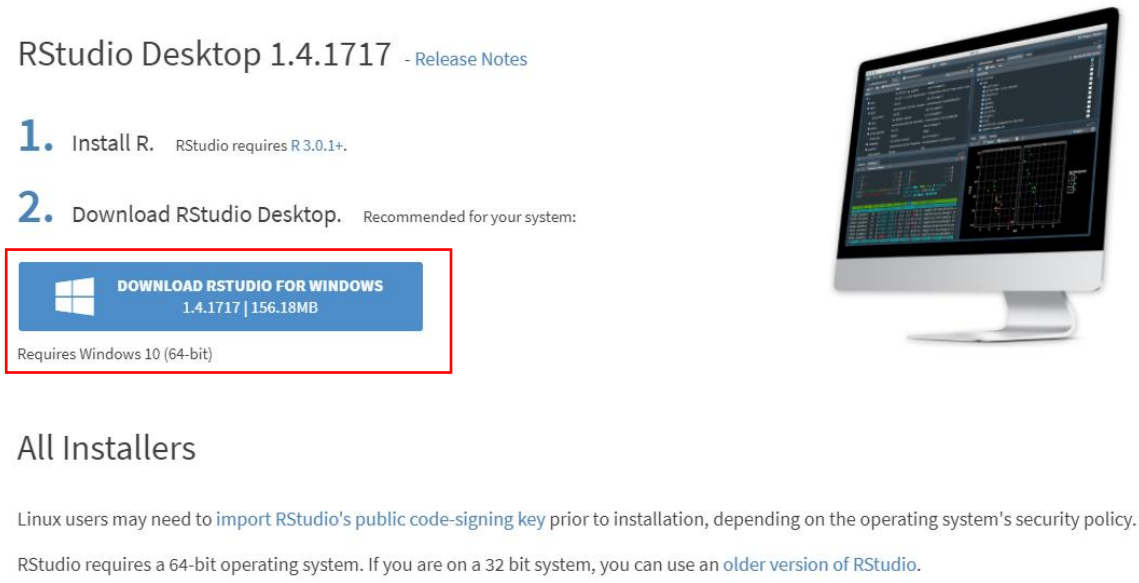


Figure 3 Download RStudio for windows.

Now right click on the downloaded file> right click and select run as administrator> it will start the setup wizard. Use the default settings. It will take some time to finish the installation process and finally click on finish button. We have successfully installed RStudio on windows 10 machine. A video of installation of R Studio can be found https://www.youtube.com/watch?v=NZxSA80lF1I&ab_channel=TechDecodeTutorials.

2. Start a R script

Start RStudio. Click the green plus button under the file ( ). Then click on the R script from the drop down options (Figure 4). It opens a R script and it is ready to write the code. Before writing code save it in an appropriate folder by clicking on the save button (Figure 5).

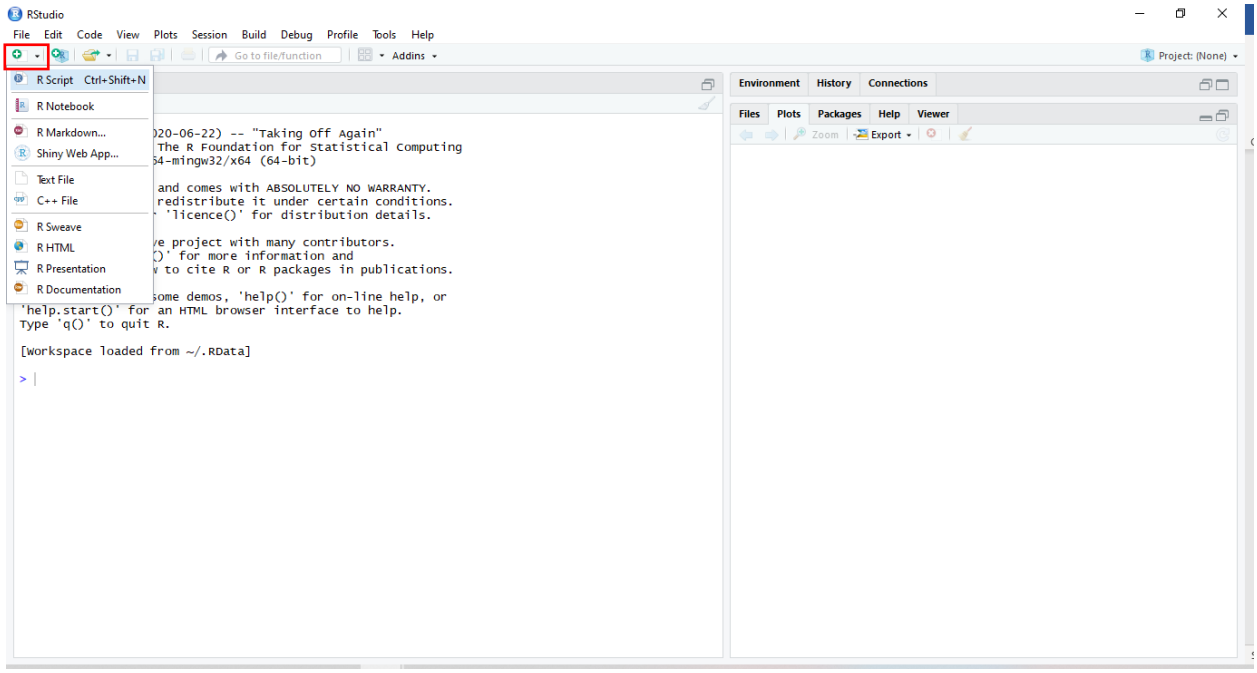


Figure 4 Start a R script.

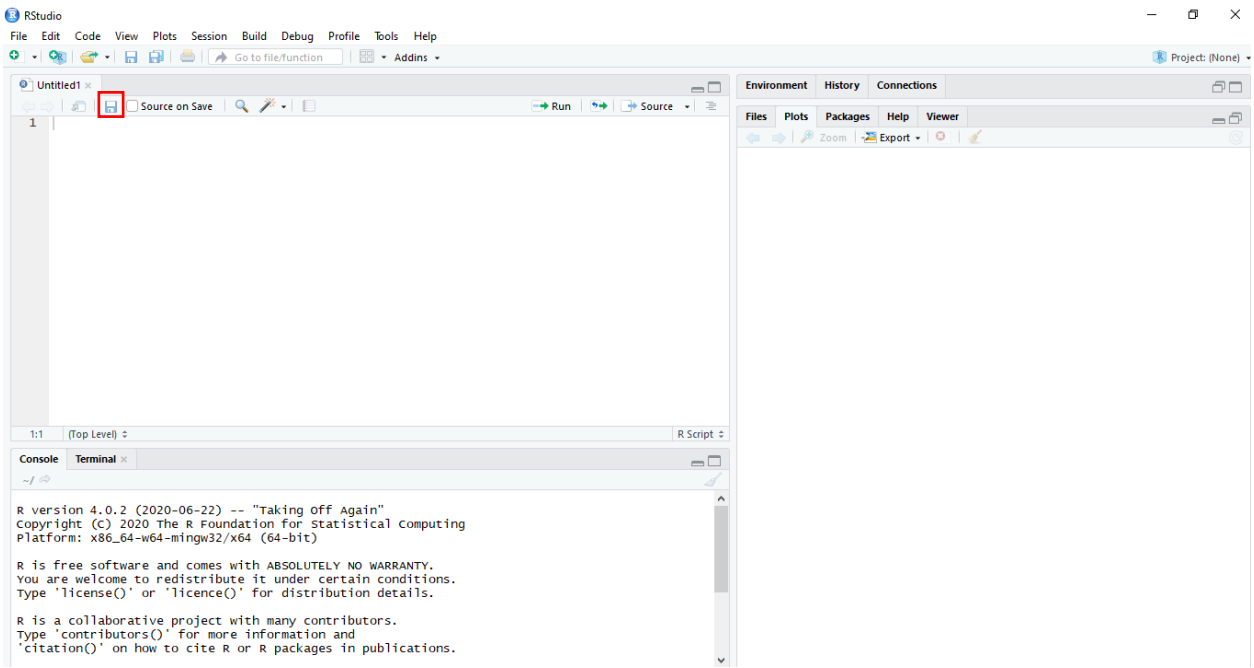


Figure 5 Save the R script.

3. Install required ML packages

To use the ML functions, we need to install the ML packages in R. Write the following code and hit the run button in R script window. It will take 1-2 minutes to install all the required packages and libraries. To run the code in R, you just need to put the cursor on the line of the code and click on the run button (Figure 6).

```
install.packages("pacman");pacman::p_load(rgdal,caret, e1071,mclust, randomForest, kernlab)
```

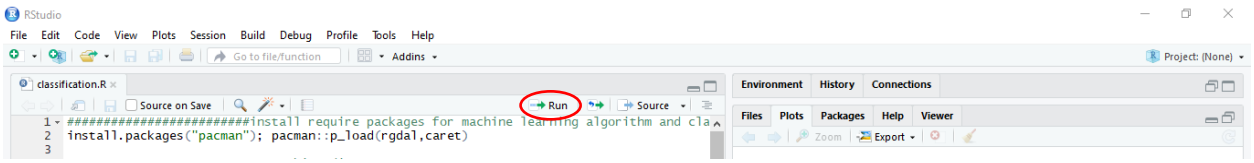


Figure 6 Install required packages for machine learning algorithm and classification.

Pacman: The pacman package is an R package management tool that combines the functionality of base library related functions into intuitively named functions. This package is ideally added to .Rprofile to increase workflow by reducing time recalling obscurely named functions, reducing code and integrating functionality of base functions to simultaneously perform multiple actions.

Caret: The caret package (short for Classification And REgression Training) contains functions to streamline the model training process for complex regression and classification problems. The package utilizes a number of R packages but tries not to load them all at package start-up (by removing formal package dependencies, the package startup time can be greatly decreased).

rgdal: Provides bindings to the 'Geospatial' Data Abstraction Library ('GDAL') and access to projection/transformation operations from the 'PROJ' library. Use is made of classes defined in the 'sp' package. Raster and vector map data can be imported into R, and raster and vector 'sp' objects exported.

4. Set working directory

code: write the following code and appropriate file path (where you save the training samples and other required data) in your ML model. Now hit the run button to set the working directory.

```
setwd("J:/SJWMD/R/R_CodeForML/Rtest/WestLANS")"
```

5. Import training samples and data to be classified

code: write the following line of code to import the training samples saved in the csv file (the name probably different than mine). Here, ‘samples’ is the variable name. You can choose a different name if you want. Hit the run button to run the code.

```
samples <- read.csv("Training.csv")
```

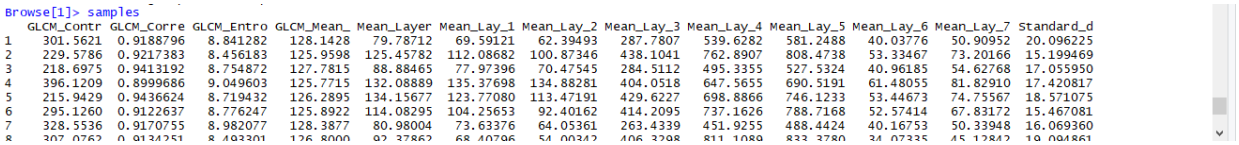
```
Data<-read.csv("InputForPrediction.csv")
```

read.csv: we use the built in read.csv(...) function which reads the data in as a data frame.

6. Visualize the training data

If you want to see how the data look like. You just need to write the name of the variable and put the cursor on it and click the run button. The following code will show the sample data in the console. **Note here, the data should have no other attributes such as FID and shape exported from ArcGIS. Only attributes used in the classification, as shown in Figure 7.**

```
samples
```



```
Browse[1]> samples
   GLCM_Contr GLCM_Corre GLCM_Entro GLCM_Mean_ Mean_Layer Mean_Lay_1 Mean_Lay_2 Mean_Lay_3 Mean_Lay_4 Mean_Lay_5 Mean_Lay_6 Mean_Lay_7 Standard_d
1    301.5621  0.9188796   8.841282   128.1428   79.78712   69.59121   62.39493   287.7807   539.6282   581.2488   40.03776   50.90952  20.096225
2    229.5786  0.9217383   8.456183   125.9598  125.45782  112.08682  100.87346   438.1041   762.8907   808.4738   53.33467   73.20166  15.199469
3    218.6975  0.9413192   8.754872   127.7815   88.88465   77.97396   70.47545   284.5112   495.3355   527.5324   40.96185   54.62768  17.055950
4    396.1209  0.8999686   9.049603   125.7715  132.08889  135.37698  134.88281   404.0518   647.5655   690.5191   61.48055   81.82910  17.420817
5    215.9429  0.9436624   8.719432   126.2895  134.15677  123.77080  113.47191   429.6227   698.8866   746.1233   53.44673   74.75567  18.571075
6    295.1260  0.9122637   8.776247   125.8922  114.08295  104.25653   92.40162   414.2095   737.1626   788.7168   52.57414   67.83172  15.467081
7    328.5536  0.9170755   8.982077   128.3877   80.98004   73.63376   64.05361   263.4339   451.9255   488.4424   40.16753   50.33948  16.069360
8    307.0762  0.9134251   8.493301   126.8000   92.37862   68.40796   54.00342   406.3298   811.1089   833.3780   34.07335   45.12842  19.094861
```

Figure 7 Training samples.

7. Get the Class column and display it

The Class (target attribute) in the training data file should be saved as the last column of the csv file. We need to know the index number of this dependent variable or the column that we need to predict. Here the dependent variable or column is 'Class'. Use the code to save this index to nIndexClass variable:

```
nIndexClass<- ncol(samples)
```

```
samples[nIndexClass]
```

8. Set up 10-fold Cross Validation tuning

Code: following code will control all the parameters to train the machine learning classifiers. The details of the arguments of this line of code are described in below.

```
# set up k-Fold to 10, cv means cross-validation
```

```
trainctrl <- trainControl(method = "cv", number = 10, allowParallel = TRUE,  verboseIter = TRUE, savePredictions = "final")
```

TrainControl: this function controls parameters in training.

Arguments in this function

Method: The resampling method: "boot", "boot632", "optimism_boot", "boot_all", "cv", "repeatedcv", "LOOCV", "LGOCV"(for repeated training/test splits),"none"(only fits one model to the entire training set),"oob"(only for random forest, bagged trees, bagged earth, bagged flexible discriminant analysis, or conditional tree forest models), timeslice, "adaptive_cv", "adaptive_boot" or "adaptive_LGOCV"

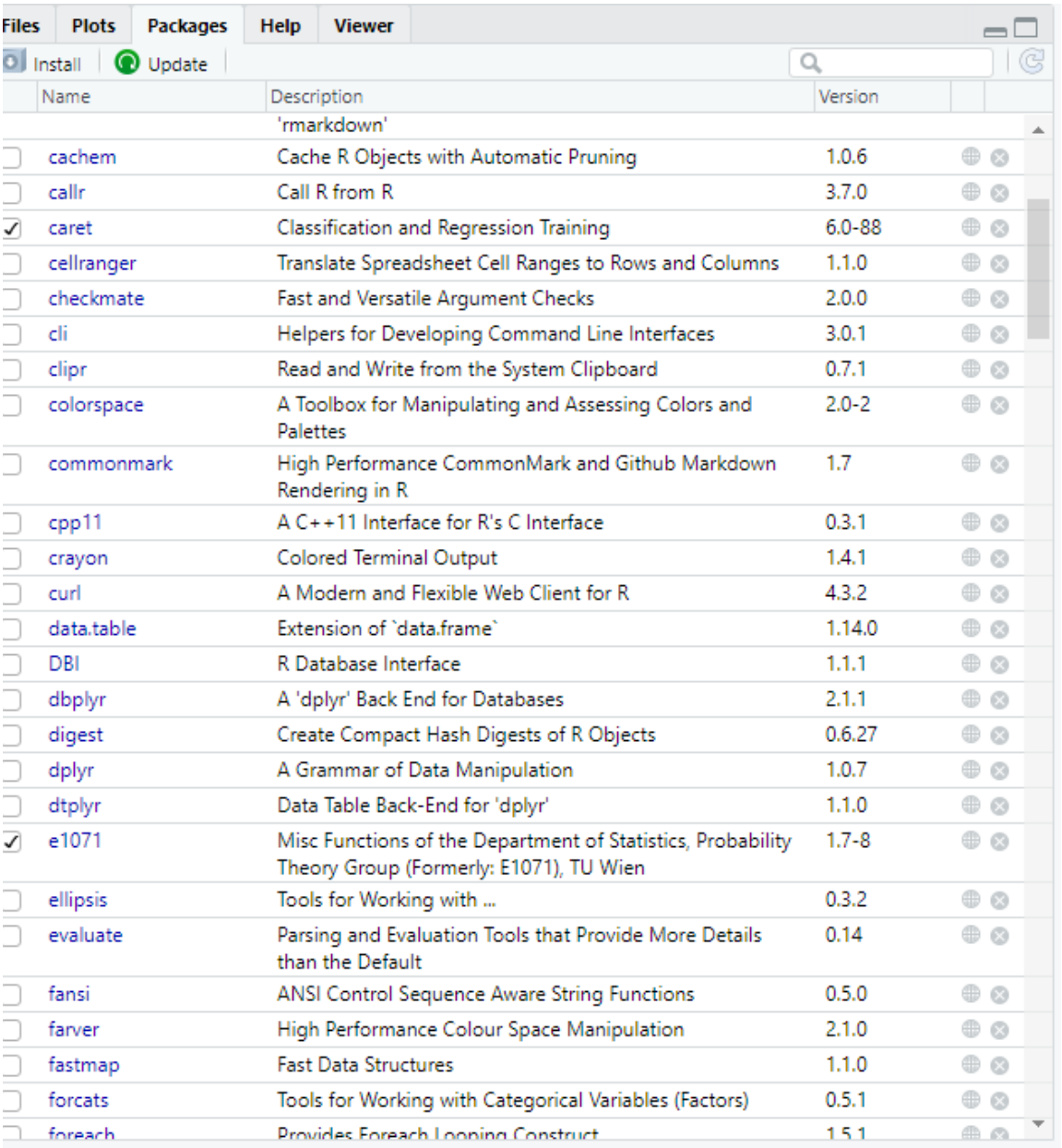
Number: Either the number of folds or number of resampling iterations

verboseIter: A logical for printing a training log.

savePredictions: an indicator of how much of the hold-out predictions for each resample should be saved. Values can be either "all", "final", or "none". A logical value can also be used that convert to "all" (for true) or "none" (for false). **"final" saves the predictions for the optimal tuning parameters.**

allowParallel: if a parallel backend is loaded and available, should the function use it.

If you get error notifying that you cannot run this trainControl function, make sure you have installed relevant packages and also in R studio check the caret, e1070 packages as shown below



9. Train ML classification models

Train a Random Forest (RF) model

The following code is used to train a Random Forest (RF) model. Here, x is the independent variables and y is the dependent variables. Independent variables (also referred to as Features) are the input for a process that is being analyzes. Dependent variables are the output of the process. For example, in our dataset (samples), the dependent variable is the Class column (the index for the class column is nIndexClass) and others are independent variables. 'y' should be in factor. That is why here we need 'as.factor' function to convert the class column into factor. Now put the cursor anywhere of this line of code and click run button. It will take 5-10 minutes finish the run. Th1e running time depends on the size of the data. You will see the progress in the console panel. The arguments are almost similar for all the classifiers. The details about the arguments are discussed below.

```
RF_model <- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "rf", ntree = 500, tuneLength = 10, preProcess = c("center", "scale"), trControl = trainctrl, metric="Kappa")
```

Train a Support Vector Machine(SVM) model

Code:

```
SVM_model <- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "svmRadial", tuneLength = 10, preProcess = c("center", "scale"),trControl = trainctrl, metric="Kappa")
```

Train k-Nearest Neighbor (kNN) model:

Code:

```
kNN_model <- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "knn",preProcess = c("center", "scale"),tuneLength = 10, trControl = trainctrl, metric="Kappa")
```

Train an Artificial Neural Network (ANN)

Code:

```
ANN_model <- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "nnet", preProcess = c("center", "scale"),tuneLength = 10, trControl = trainctrl, metric="Kappa")
```

Train: This function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure.

x: data frame containing training data where samples are in rows and features are in columns.

y: a numeric or factor vector containing the outcome for each sample.

Method: a string specifying which classification or regression model to use. Possible values are: ada, bag, bagEarth, bagFDA, blackboost, cforest, ctree, ctree2 arguments passed to the classification or regression routine (such as randomForest). Errors will occur if values for tuning parameters are passed here.

ntree: In the random forests literature, this is referred to as the ntree parameter. Larger number of trees produce more stable models and covariate importance estimates, but require more memory and a longer run time. For small datasets, 50 trees may be sufficient. For larger datasets, 500 or more may be required.

tuneLength: an integer denoting the number of levels for each tuning parameters that should be generated by createGrid. (NOTE: If given, this argument must be named.)

preProcess: Pre-processing transformation (centering, scaling etc.) can be estimated from the training data and applied to any data set with the same variables.

Center: The value of center determines how column centering is performed. If center is a numeric-alike vector with length equal to the number of columns of x, then each column of x has the corresponding value from center subtracted from it. If center is TRUE then centering is done by subtracting the column means (omitting NAs) of x from their corresponding columns, and if center is FALSE, no centering is done.

Scale: The value of scale determines how column scaling is performed (after centering). If scale is a numeric-alike vector with length equal to the number of columns of x, then each column of x is divided by the corresponding value from scale. If scale is TRUE then scaling is done by dividing the (centered) columns of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done.

trControl: a list of values that define how this function acts.

metric: a string that specifies what summary metric will be used to select the optimal model. By default, possible values are "RMSE" and "Rsquared" for regression and "Accuracy" and "Kappa" for classification.

10. Generate error matrix

In R, the error matrix, a specific table layout that allows visualization of the performance of an algorithm or model. It is a table that will categorize the predictions against the actual values. It includes two dimensions, among them one will indicate the predicted values and another one will represent the actual values. Each row in the confusion matrix will represent the predicted values and columns will be responsible for actual values. This can also be vice-versa. The following code will create confusion matrix or error matrix for each classifier.

Code:

```
RF_errorM <- confusionMatrix(RF_model$pred$obs,RF_model$pred$pred)
```

```
SVM_errorM <-confusionMatrix(SVM_model$pred$obs,SVM_model$pred$pred)

kNN_errorM <-confusionMatrix(kNN_model$pred$obs,kNN_model$pred$pred)

ANN_errorM <-confusionMatrix(ANN_model$pred$obs,ANN_model$pred$pred)
```

cofusionMatrix: Calculates a cross-tabulation of observed and predicted classes with associated statistics.

RF_model$pred$obs: observed classes.

RF_model$pred$pred: prediction values or classes for the classifier.

11. Export error matrix to a text file

We can export the error matrix of the training model and copy/paste it to a text file, and then revise it in Excel. Following codes generate the error matrix in the console panel for the each of the models. Figure 8 shows the printed error matrix for Random Forest model. We can simply copy the output and paste it to a new text file and save it.

```
RF_errorM

SVM_errorM

kNN_errorM

ANN_errorM
```

```
Browse[1]> rf_errorM
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
        1  179   0   0   0   0   0   0   0   0   8   0   6   0   0   1   0   0   0
        2    0 122   0   0   0   0  34   0   0   1   0   0   0   0   1   0   0  27
        3    0   0   9   0   0   0   0   0   0   5   0   0   0   0   0   0   0   0
        4    0   2   0   7   2   0   2   0  24   0   0   0   0   0   1   0   0   1
        5    0   1   0   1  50   0   0   0   0   0  26   0   0   0   2   0   0   0
        6    3   0   0   0   0   6   0   0   0   0   0   2   0   0   3   1   0   0
        7    0  32   0   0   0   0 164   0   4   1   2   0   0   0   3   0   0  31
        8    0   0   0   0   0   0   0 562   0   0   1   0   0   0  65   2   0   0
        9    0   2   0   1   1   0   6   1 243   0   4   0   0   0  16   0   0   3
        10   5   0   1   0   0   0   1   1   0 126   0   0   0   0   0   0   0   0
        11   0   0   0   0  14   0   0   0   7   0 244   0   0  19   4   0   0   0
        12  12   0   0   0   0   0   3   2   0   0   0  51   0   0   1  10   0   0
        13   6   0   0   0   0   0   0   0   0   0   0   1  20   0   1  10   0   0
        14   0   0   0   0   0   0   0   1   0   0  12   0   0  83   1   0   0   0
        15   0   1   0   0   0   0   3 102  25   1   1   2   0   0 413   5   0   3
        16   0   0   1   0   0   2   0   7   0   1   0   0   8   0   6  86   0   0
        17   0   0   0   0   2   0   0   1   1   0   0   0   0   0  26   2   2   0
        18   0  29   0   0   0   0  29   0   3   0   0   0   0   0  10   0   0 101

Overall Statistics

               Accuracy : 0.7761
                 95% CI : (0.7612, 0.7905)
    No Information Rate : 0.2129
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7483

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12 Class: 13 Class: 14
Sensitivity           0.87317  0.64550 0.818182 0.777778  0.72464 0.750000  0.67769   0.8301  0.79153   0.88112   0.84138   0.82258  0.714286   0.81373
Specificity           0.99496  0.97894 0.998422 0.989909  0.99036 0.997163  0.97515   0.9728  0.98817   0.99737   0.98478   0.99102  0.994289   0.99545
Pos Pred Value        0.92268  0.65946 0.642857 0.179487  0.62500 0.400000  0.69198   0.8921  0.87726   0.94030   0.84722   0.64557  0.526316   0.85567
Neg Pred Value        0.99129  0.97763 0.999368 0.999363  0.99387 0.999368  0.97350   0.9549  0.97795   0.99442   0.98409   0.99645  0.997454   0.99384
Prevalence            0.06447  0.05943 0.003459 0.002830  0.02170 0.002516  0.07610   0.2129  0.09654   0.04497   0.09119   0.01950  0.008805   0.03208
Detection Rate        0.05629  0.03836 0.002830 0.002201  0.01572 0.001887  0.05157   0.1767  0.07642   0.03962   0.07673   0.01604  0.006289   0.02610
Detection Prevalence  0.06101  0.05818 0.004403 0.012264  0.02516 0.004717  0.07453   0.1981  0.08711   0.04214   0.09057   0.02484  0.011950   0.03050
Balanced Accuracy     0.93406  0.81222 0.908302 0.883843  0.85750 0.873581  0.82642   0.9015  0.88985   0.93924   0.91308   0.90680  0.854288   0.90459
                     Class: 15 Class: 16 Class: 17 Class: 18
Sensitivity             0.7455   0.74138 1.0000000   0.60843
Specificity             0.9455   0.99184 0.9899308   0.97644
Pos Pred Value          0.7428   0.77477 0.0588235   0.58721
Neg Pred Value          0.9463   0.99022 1.0000000   0.97839
Prevalence              0.1742   0.03648 0.0006289   0.05220
Detection Rate          0.1299   0.02704 0.0006289   0.03176
Detection Prevalence    0.1748   0.03491 0.0106918   0.05409
Balanced Accuracy       0.8455   0.86661 0.9949654   0.79244
```

Figure 8 A sample of error matrix of Random Forest model output in R Console.

12. Export error matrix as a csv file

We can also export the error matrix as a csv file. After running the following codes, error matrix of each classifier will be saved in the working directory we set at the beginning.

```
#Export the error matrix as CSV files
cm_RF <- as.table(RF_errorM)
write.csv(cm_RF,"RF_ErrorMatrix.csv")
cm_SVM <- as.table(SVM_errorM)
write.csv(cm_SVM,"SVM_ErrorMatrix.csv")
cm_kNN <- as.table(kNN_errorM)
write.csv(cm_kNN,"kNN_ErrorMatrix.csv")
cm_ANN <- as.table(ANN_errorM)
write.csv(cm_ANN,"Ann_ErrorMatrix.csv")
```

13. Edit the error matrix in Excel

Figure 9 shows how it looks like in csv file. We can add the Overall Accuracy (OA), Kappa, Producer's Accuracy (PA) and User's Accuracy (UA) in excel from the R outputs (Figure 10). We will find all of these statistics in error matrix text files.

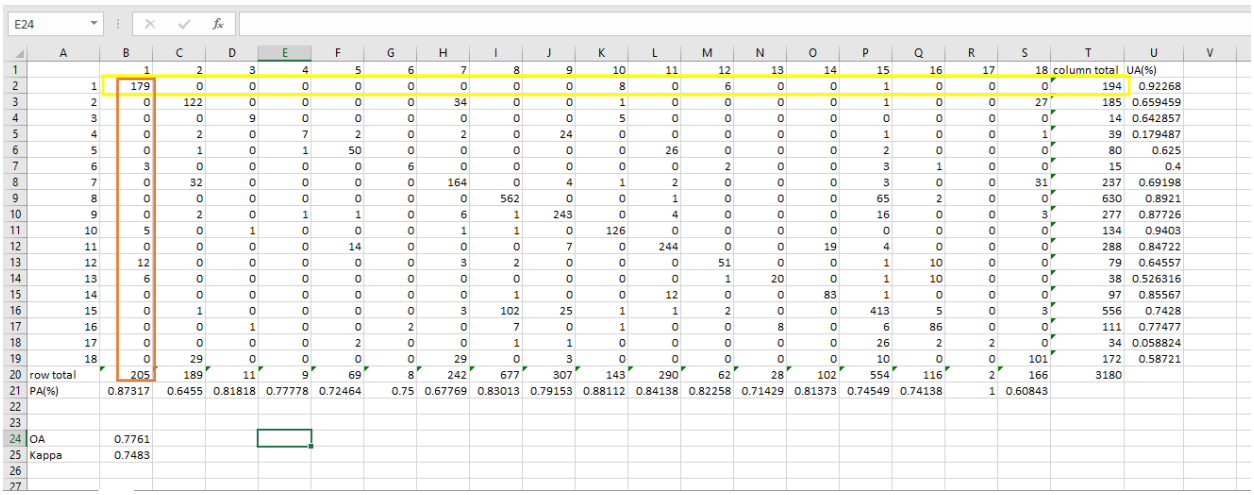
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	179	0	0	0	0	0	0	0	0	8	0	6	0	0	1	0	0	0
2	0	122	0	0	0	0	34	0	0	1	0	0	0	0	1	0	0	27
3	0	0	9	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
4	0	2	0	7	2	0	2	0	24	0	0	0	0	0	1	0	0	1
5	0	1	0	1	50	0	0	0	0	0	26	0	0	0	2	0	0	0
6	3	0	0	0	0	6	0	0	0	0	0	2	0	0	3	1	0	0
7	0	32	0	0	0	0	164	0	4	1	2	0	0	0	3	0	0	31
8	0	0	0	0	0	0	0	562	0	0	1	0	0	0	65	2	0	0
9	0	2	0	1	1	0	6	1	243	0	4	0	0	0	16	0	0	3
10	5	0	1	0	0	0	1	1	0	126	0	0	0	0	0	0	0	0
11	0	0	0	0	14	0	0	0	7	0	244	0	0	19	4	0	0	0
12	12	0	0	0	0	0	3	2	0	0	0	51	0	0	1	10	0	0
13	6	0	0	0	0	0	0	0	0	0	0	1	20	0	1	10	0	0
14	0	0	0	0	0	0	0	1	0	0	12	0	0	83	1	0	0	0
15	0	1	0	0	0	0	3	102	25	1	1	2	0	0	413	5	0	3
16	0	0	1	0	0	2	0	7	0	1	0	0	8	0	6	86	0	0
17	0	0	0	0	2	0	0	1	1	0	0	0	0	0	26	2	2	0
18	0	29	0	0	0	0	29	0	3	0	0	0	0	0	10	0	0	101

Figure 9 The error matrix for RF model in CSV file.

We can open a new excel file and name it 'ErrorMatrix'. Copy the error matrix of Random Forest (RF) to the excel sheet. Add a column and named it "column total". Use sum equation to get the values for this column. Add a row and named it "row total". Use the sum equation to get the values for this row. Add another column and named it "UA(%)" and fill it with values 'pos pred value' from the error matrix we printed in R for the random forest. Add another row and named it "PA(%)". Fill it with values 'sensitivity' from the error matrix for random forest. Add the overall accuracy (OA) and kappa.

```
Overall Statistics

               Accuracy : 0.7761
                 95% CI : (0.7612, 0.7905)
    No Information Rate : 0.2129
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7483

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9 Class: 10 Class: 11 Class: 12 Class: 13 Class: 14
Sensitivity           0.87317  0.64550 0.818182 0.777778  0.72464 0.750000  0.67769   0.8301  0.79153  0.88112  0.84138  0.82258 0.714286  0.81373
Specificity           0.99496  0.97894 0.998422 0.989909  0.99036 0.997163  0.97515   0.9728  0.98817  0.99737  0.98478  0.99102 0.994289  0.99545
Pos Pred Value        0.92268  0.65946 0.642857 0.179487  0.62500 0.400000  0.69198   0.8921  0.87726  0.94030  0.84722  0.64557 0.526316  0.85567
Neg Pred Value        0.99129  0.97763 0.999368 0.999363  0.99387 0.999368  0.97350   0.9549  0.97795  0.99442  0.98409  0.99645 0.997454  0.99384
Prevalence            0.06447  0.05943 0.003459 0.002830  0.02170 0.002516  0.07610   0.2129  0.09654  0.04497  0.09119  0.01950 0.008805  0.03208
Detection Rate        0.05629  0.03836 0.002830 0.002201  0.01572 0.001887  0.05157   0.1767  0.07642  0.03962  0.07673  0.01604 0.006289  0.02610
Detection Prevalence  0.06101  0.05818 0.004403 0.012264  0.02516 0.004717  0.07453   0.1981  0.08711  0.04214  0.09057  0.02484 0.011950  0.03050
Balanced Accuracy     0.93406  0.81222 0.908302 0.883843  0.85750 0.873581  0.82642   0.9015  0.88985  0.93924  0.91308  0.90680 0.854288  0.90459
                     Class: 15 Class: 16 Class: 17 Class: 18
Sensitivity             0.7455   0.74138 1.0000000  0.60843
Specificity             0.9455   0.99184 0.9899308  0.97644
Pos Pred Value          0.7428   0.77477 0.0588235  0.58721
Neg Pred Value          0.9463   0.99022 1.0000000  0.97839
Prevalence              0.1742   0.03648 0.0006289  0.05220
Detection Rate          0.1299   0.02704 0.0006289  0.03176
Detection Prevalence    0.1748   0.03491 0.0106918  0.05409
Balanced Accuracy       0.8455   0.86661 0.9949654  0.79244
```

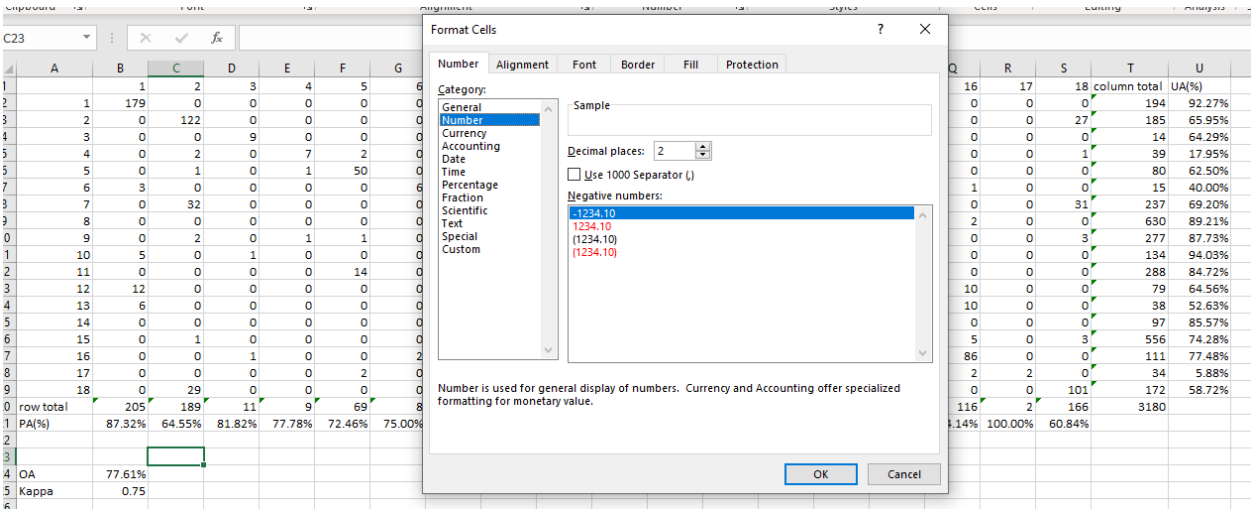
Figure 10 Printed error matrix for Random Forest model. PA = sensitivity and UA = pos pred value.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	column total	UA(%)	
2	1	179	0	0	0	0	0	0	0	0	8	0	6	0	0	1	0	0	0	194	0.92268	
3	2	0	122	0	0	0	0	34	0	0	1	0	0	0	0	1	0	0	27	185	0.659459	
4	3	0	0	9	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	14	0.642857	
5	4	0	2	0	7	2	0	2	0	24	0	0	0	0	0	1	0	0	1	39	0.179487	
6	5	0	1	0	1	50	0	0	0	0	0	26	0	0	0	2	0	0	0	80	0.625	
7	6	3	0	0	0	0	6	0	0	0	0	0	2	0	0	3	1	0	0	15	0.4	
8	7	0	32	0	0	0	0	164	0	4	1	2	0	0	0	3	0	0	31	237	0.69198	
9	8	0	0	0	0	0	0	0	562	0	0	1	0	0	0	65	2	0	0	630	0.8921	
10	9	0	2	0	1	1	0	6	1	243	0	4	0	0	0	16	0	0	3	277	0.87726	
11	10	5	0	1	0	0	0	1	1	0	126	0	0	0	0	0	0	0	0	134	0.9403	
12	11	0	0	0	0	14	0	0	0	7	0	244	0	0	19	4	0	0	0	288	0.84722	
13	12	12	0	0	0	0	0	3	2	0	0	0	51	0	0	1	10	0	0	79	0.64557	
14	13	6	0	0	0	0	0	0	0	0	0	0	1	20	0	1	10	0	0	38	0.526316	
15	14	0	0	0	0	0	0	0	1	0	0	12	0	0	83	1	0	0	0	97	0.85567	
16	15	0	1	0	0	0	0	3	102	25	1	1	2	0	0	413	5	0	3	556	0.7428	
17	16	0	0	1	0	0	2	0	7	0	1	0	0	8	0	6	86	0	0	111	0.77477	
18	17	0	0	0	0	2	0	0	1	1	0	0	0	0	0	26	2	2	0	34	0.058824	
19	18	0	29	0	0	0	0	29	0	3	0	0	0	0	0	10	0	0	101	172	0.58721	
20	row total	205	189	11	9	69	8	242	677	307	143	290	62	28	102	554	116	2	166	3180		
21	PA(%)	0.87317	0.6455	0.81818	0.77778	0.72464	0.75	0.67769	0.83013	0.79153	0.88112	0.84138	0.82258	0.71429	0.81373	0.74549	0.74138	1	0.60843			
22																						
23																						
24	OA	0.7761																				
25	Kappa	0.7483																				
26																						
27																						

Figure 11 Error matrix in excel file.

We can configure the UA and PA to percentage as shown in Figure 12.



Open a new sheet. Name it SVM and make it similar to the RF.

Figure 12 Final error matrix in excel for RF model.

14. McNemar tests

McNemar tests can tell whether the classification between two classifiers is significant or not. The following codes are used to conduct McNemar tests.

```
# Conduct McNemar Test
# Define the McNemar function
# Added by Caiyun Zhang
```

```
McNemar_Test<-function(Actual, Class1, Class2, nTotal) {

  f11<-0
  f12<-0
  f21<-0
  f22<-0
  z_Value<-0
  for (i in 1:nTotal) {
    if (Actual[i]==Class1[i] && Actual==Class2[i]) {
      f11<-f11+1
    }
    if (Actual[i]==Class1[i] && Actual[i]!=Class2[i]) {
      f12<-f12+1
    }
    if (Actual[i]!=Class1[i] && Actual[i]==Class2[i]) {
      f21<-f21+1
    }
    if (Actual[i]!=Class1[i] && Actual[i]!=Class2[i]) {
      f22<-f22+1
    }
  }
  z_Value<-abs((f12-f21)/sqrt(f12+f21))
  return(round(z_Value, digits=2))
}

# Call the McNemar function
nTotalRecord<-nrow(samples)
zValue_ANN_SVM<-McNemar_Test(class_actual,class_ANN,class_SVM,nTotalRecord)
zValue_ANN_kNN<-McNemar_Test(class_actual,class_ANN,class_kNN,nTotalRecord)
zValue_ANN_RF<-McNemar_Test(class_actual,class_ANN,class_RF,nTotalRecord)
zValue_SVM_kNN<-McNemar_Test(class_actual,class_SVM,class_kNN,nTotalRecord)
zValue_SVM_RF<-McNemar_Test(class_actual,class_SVM,class_RF,nTotalRecord)
zValue_RF_kNN<-McNemar_Test(class_actual,class_RF,class_kNN,nTotalRecord)
# write out the McNemar test results in csv file
tests<-c("ANN/SVM", "ANN/kNN", "ANN/RF", "SVM/kNN", "SVM/RF", "RF/kNN")
zValues<-c(zValue_ANN_SVM,
zValue_ANN_kNN,zValue_ANN_RF,zValue_SVM_kNN,zValue_SVM_RF,zValue_RF_kNN)
finalOut<-cbind(tests, zValues)
write.csv(finalOut,"McNemarTest.csv")
```

15. Ensemble analysis and export error matrix as csv files

The following codes are used to conduct ensemble analysis of RF, SVM, and ANN, and generate error matrix of the ensemble analysis results, as well as RF, SVM, ANN, and kNN.

```
# Conduct ensemble analysis classification and error matrix
# Write out error matrix of RF, SVM, ANN and kNN classifiers
# We only combine three classifications from ANN, SVM, and RF
# Coded and added by Caiyun Zhang
nMaxClass<-max(samples$Class) # get the maximum of class number code
nMinClass<-min(samples$Class) # get the minimum of class number code
RF_UA<-0  # define User's accuracy of RF
SVM_UA<-0 # define User's accuracy of SVM
ANN_UA<-0 # define User's accuracy of ANN
kNN_UA<-0 # define User's accuracy of kNN

RF_PA<-0  # define producer's accuracy of RF
SVM_PA<-0 # define producer's accuracy of SVM
ANN_PA<-0 # define producer's accuracy of ANN
kNN_PA<-0 # define producer's accuracy of kNN

# Get the error matrix table
cm_RF <- as.table(RF_errorM)
cm_SVM <- as.table(SVM_errorM)
cm_kNN <- as.table(kNN_errorM)
cm_ANN <- as.table(ANN_errorM)
# Calculate user's accuracy, and producer's accuracy, row total, and column total
ColSum_RF<-rowSums(cm_RF)
ColSum_SVM<-rowSums(cm_SVM)
ColSum_ANN<-rowSums(cm_ANN)
ColSum_kNN<-rowSums(cm_kNN)

RowSum_RF<-colSums(cm_RF)
RowSum_SVM<-colSums(cm_SVM)
RowSum_ANN<-colSums(cm_ANN)
RowSum_kNN<-colSums(cm_kNN)

RF_PA_UA<-RF_errorM$byClass
SVM_PA_UA<-SVM_errorM$byClass
ANN_PA_UA<-ANN_errorM$byClass
kNN_PA_UA<-kNN_errorM$byClass

for (i in nMinClass:nMaxClass) {

  RF_UA[i]<-RF_PA_UA[i,3] # the third column is UA
  SVM_UA[i]<-SVM_PA_UA[i,3]
  ANN_UA[i]<-ANN_PA_UA[i,3]
  kNN_UA[i]<-kNN_PA_UA[i,3]

  RF_PA[i]<-RF_PA_UA[i,1] # the first column is PA
  SVM_PA[i]<-SVM_PA_UA[i,1]
  ANN_PA[i]<-ANN_PA_UA[i,1]
  kNN_PA[i]<-kNN_PA_UA[i,1]

}
```

```
nTotalRowReference<-nrow(samples)
nTotalRowReference
class_EA<-0 # define ensemble analysis prediction
nMajorityVote<-0 # define the majority vote variable
UA_Max<-0 # define the maximum user's accuracy among three votes

for (i in 1:nTotalRowReference) {
  x<-c(class_RF[i], class_SVM[i], class_ANN[i])
  nMajorityVote<-majorityVote(x)
  UA_Max<-0
  # if a majority vote is achieved
  if (max(nMajorityVote$table)!=1)
  {
    class_EA[i]<-nMajorityVote$majority

  }
  # if no agreement is achieved
  else {
    if (RF_UA[class_RF[i]]>UA_Max) {
      class_EA[i]<-class_RF[i]
      UA_Max<-RF_UA[class_RF[i]]
    }
    if (SVM_UA[class_SVM[i]]>UA_Max) {
      class_EA[i]<-class_SVM[i]
      UA_Max<-SVM_UA[class_SVM[i]]
    }
    if (ANN_UA[class_ANN[i]]>UA_Max) {
      class_EA[i]<-class_ANN[i]
    }
    # to test which record has different votes from three classifiers
    print (i)
  }
}

# Print out all training sample prediction (optional)
 all_pred <- cbind(class_actual,class_RF, class_SVM, class_ANN, class_kNN, class_EA)
 write.csv(all_pred, "all_pred.csv", row.names = FALSE)
# Now read in the prediction. This is weird but this can order the EA error matrix level in the same order
as other classifiers
 predModel <- read.csv("all_pred.csv")
 EA_errorM<- confusionMatrix(as.factor(predModel$class_actual),as.factor(predModel$class_EA))
 EA_errorM

 # calculate UA and PA of ensemble analysis, and all classifiers out as a csv file
 cm_EA <- as.table(EA_errorM)
 ColSum_EA<-rowSums(cm_EA)
 RowSum_EA<-colSums(cm_EA)
 EA_UA<-0
 EA_PA<-0
 for (i in nMinClass:nMaxClass) {
```

```
  EA_UA[i]<-cm_EA[i,i]/ColSum_EA[i]
  EA_PA[i]<-cm_EA[i,i]/RowSum_EA[i]
}

# write out csv error matrix function
WriteErrorM<-function(EM,ColTotal,UA,RowTotal,PA,TotalClass,TotalR,FileName) {

  Accuracy<-EM$overall # Get the confusion matrix's accuracy
  OA<-round(Accuracy[1],digits = 2) # Get the overall accuracy
  Kappa<-round(Accuracy[2],digits = 2) # Get the Kappa value
  ETable<-EM$table # Get the error matrix table
  UA<-round(UA,digits = 2) # format User's accuracy, 2 decimal
  PA<-round(PA,digits = 2) # format producer's accuracy. 2 decimal
  RowTotal[TotalClass+1]<-TotalR
  RowTotal[TotalClass+2]<-0
  PA[TotalClass+1]<-0
  PA[TotalClass+2]<-0
  OA[2:TotalClass+2]<-0
  Kappa[2:TotalClass+2]<-0
  AddCol<-cbind(ETable,ColTotal,UA)
  AddRow<-rbind(AddCol,RowTotal,PA, OA, Kappa)
  write.csv(AddRow,FileName)
}

# call the write out error matrix function
WriteErrorM(RF_errorM,ColSum_RF,RF_UA,RowSum_RF,RF_PA,nMaxClass, nTotalRowReference,
"RF_EM.csv")
WriteErrorM(SVM_errorM,ColSum_SVM,SVM_UA,RowSum_SVM,SVM_PA,nMaxClass,
nTotalRowReference, "SVM_EM.csv")
WriteErrorM(ANN_errorM,ColSum_ANN,ANN_UA,RowSum_ANN,ANN_PA,nMaxClass,
nTotalRowReference, "ANN_EM.csv")
WriteErrorM(kNN_errorM,ColSum_kNN,kNN_UA,RowSum_kNN,kNN_PA,nMaxClass,
nTotalRowReference, "kNN_EM.csv")
WriteErrorM(EA_errorM,ColSum_EA,EA_UA,RowSum_EA,EA_PA,nMaxClass,nTotalRowReference,
"EA_EM.csv")
```

The error matrix text file of the ensemble analysis is displayed in Figure 13.

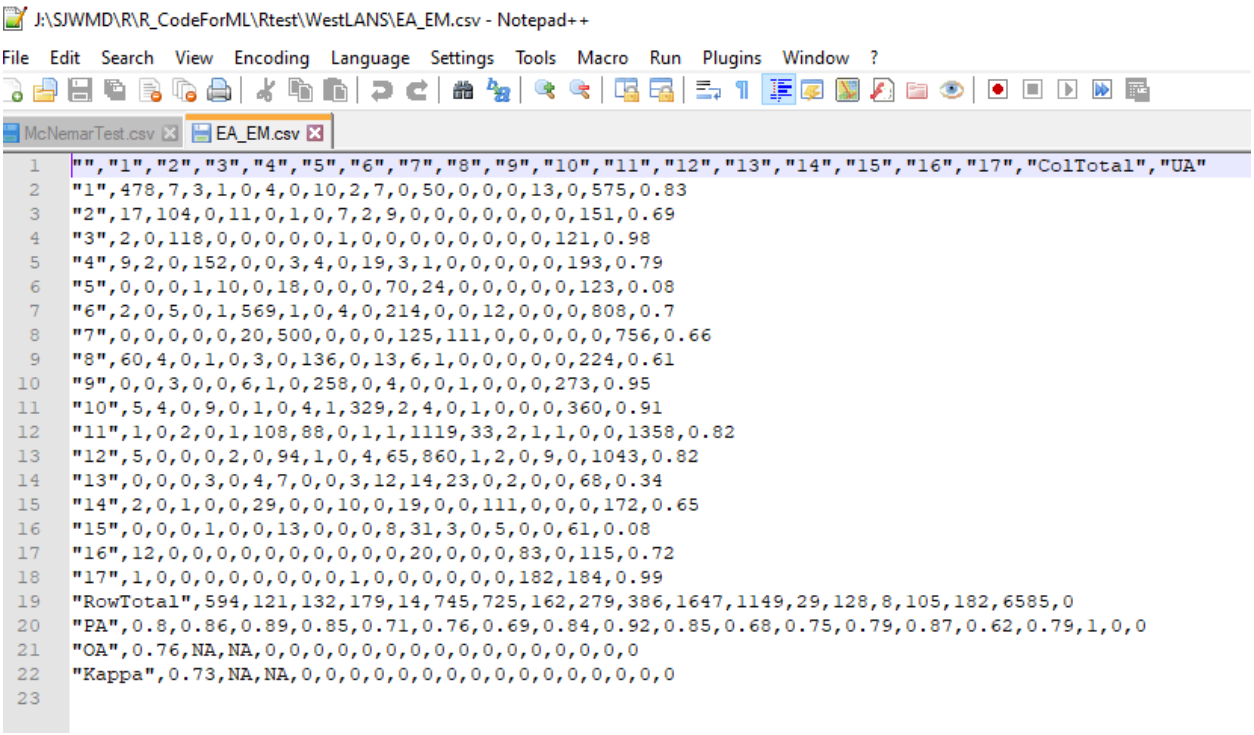


Figure 13 Output the error matrix of the ensemble analysis from R script.

16. Final prediction and ensemble prediction

After training the model, we will apply the model for final prediction to unseen dataset which is the csv file exported from ArcGIS with all features consistent with the training samples. We can include a default value of 1 to the Class target. The model will predict the target value for Class. The following codes will generate the predictions and ensemble prediction.

```
# Generate final prediction for the input data
# Load in the data for prediction/classification
# Here you need to revise to your own input data for final prediction
# this data set must have same features as the training data and last target column can be set to a default value of 1
# Added and revised by Caiyun Zhang
nRowTotal<-nrow(Data) # get the total number of objects/rows of the input data set
nRowTotal
# Generate prediction from each classifier
pred_RF = predict(RF_model,newdata = Data[-nIndexClass])
pred_SVM = predict(SVM_model,newdata = Data[-nIndexClass])
pred_kNN = predict(kNN_model,newdata = Data[-nIndexClass])
pred_ANN = predict(ANN_model,newdata = Data[-nIndexClass])

# Write out the prediction to csv file
# the output of ID sequence number can be used to join with the FID in ArcGIS shapefile for final mapping
```

```
ID<-0:(nRowTotal-1)
# Generate ensemble prediction by combining ANN, SVM, and RF predictions
pred_EA<-0 # define ensemble analysis prediction
nMajorityVote<-0 # define the majority vote variable
UA_Max<-0 # define the maximum user's accuracy among three votes

for (i in 1:nRowTotal) {
  x<-c(pred_RF[i], pred_SVM[i], pred_ANN[i])
  nMajorityVote<-majorityVote(x)
  UA_Max<-0
  # if a majority vote is achieved
  if (max(nMajorityVote$table)!=1)
  {
    pred_EA[i]<-nMajorityVote$majority

  }
  # if no agreement is achieved, three classifiers have completely different prediction
  else {
    if (RF_UA[pred_RF[i]]>UA_Max) {
      pred_EA[i]<-pred_RF[i]
      UA_Max<-RF_UA[pred_RF[i]]
    }
    if (SVM_UA[pred_SVM[i]]>UA_Max) {
      pred_EA[i]<-pred_SVM[i]
      UA_Max<-SVM_UA[pred_SVM[i]]
    }
    if (ANN_UA[pred_ANN[i]]>UA_Max) {
      pred_EA[i]<-pred_ANN[i]
    }
  }
}
EA<-as.numeric(pred_EA)
final_pred <- cbind(ID,pred_RF,pred_SVM,pred_ANN,pred_kNN,EA)
write.csv(final_pred, "FinalPrediction.csv", row.names = FALSE)
# Game over
```

A screenshot of the final prediction is shown in Figure 14. Note the code adds an “ID” field which will be used to join with the FID of the segmentation file in ArcGIS for final mapping.

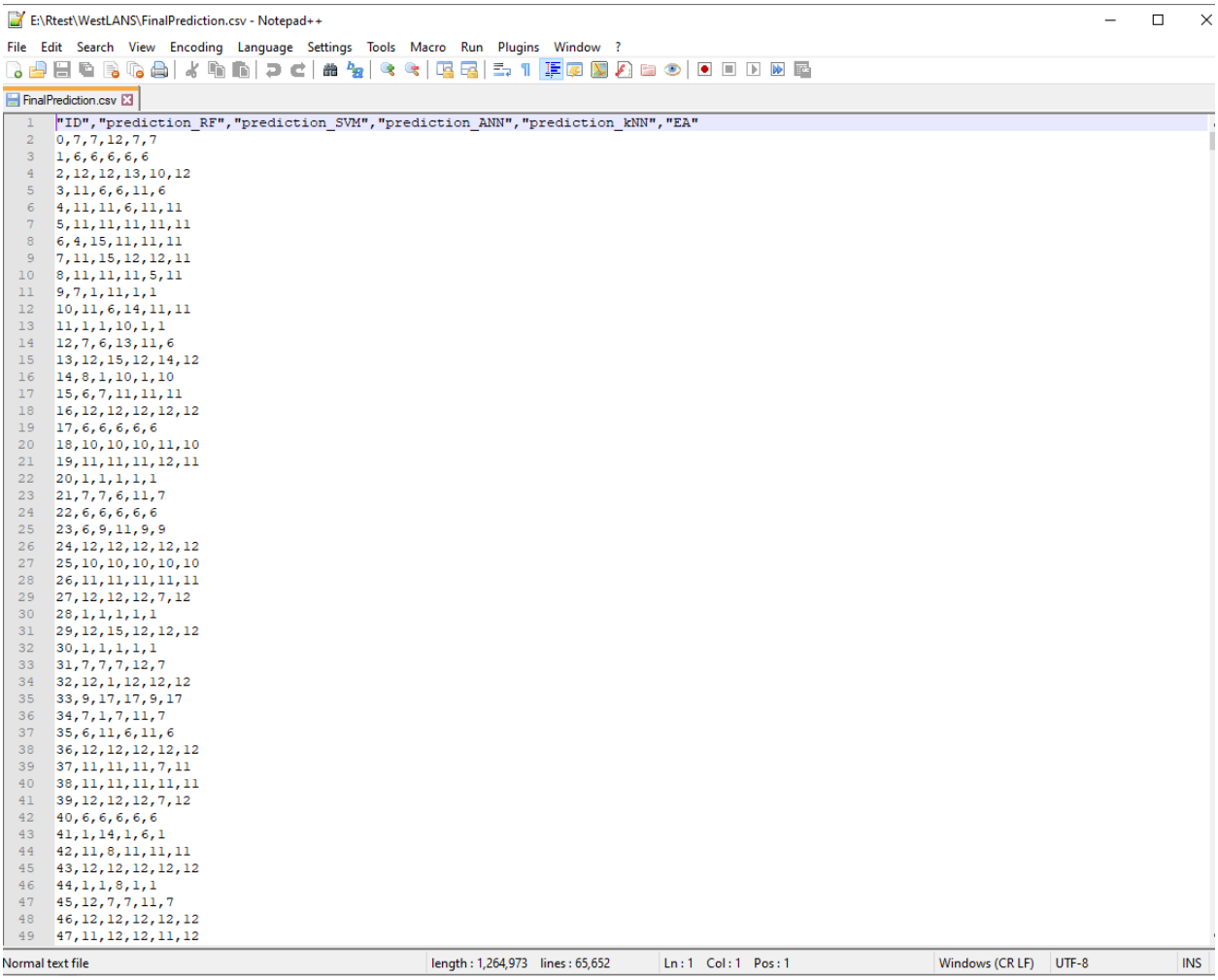


Figure 14 The final prediction file.

The whole R code is provided below, and corresponding ML_Zhang_V5.R file is also provided, which can be directly loaded in R studio. All you need is to revise the working folder, and set up the input training csv file, and csv file for final prediction. Both csv files are exported from ArcGIS.

```
##############################################################################
# R script to run Machine Learning (ML) Classifiers
# RF: Random Forest
# SVM: Support Vector Machine
# ANN: Artificial Neural Network
# kNN: k Nearest Neighbor
# Coded by: Caiyun Zhang
# Tested by: Mizanur Rhman
# Department of Geosciences, Florida Atlantic University
# Test date: 9/17/2021
# Updated date: 9/24/2021
##############################################################################
# Install required libraries and packages in R to run ML and other relevant functions
# Ensure these packages are checked
install.packages("pacman");pacman::p_load(rgdal,caret,e1071,mclust, randomForest, kernlab)
##############################################################################
# Load in required training data in csv file format
# Set up the working path/directory, must change to your own data directory
setwd("J:/SJWMD/R/R_CodeForML/Rtest/WestLANS")

# Import the training samples saved in csv file.
# Must change to your own training sample data file name
# The last column must be the Class target attribute.
# Both training data and data to be classfied must have same attributes
samples <- read.csv("Training.csv") # read in training samples, this file must be saved in the directory you set
Data<-read.csv("InputForPrediction.csv") # read in data to be predict/classified, this file must be saved in the
directory you set
# Display the data (optional)
samples
# Get the total column of the training data, and the last column is the Class variable
nIndexClass<- ncol(samples)
# Display the class column (optional)
samples[nIndexClass]
##############################################################################

# Set up k-fold cross validation
# Here we set up 10 fold. This number can be changed by revising the number in the function
trainctrl<- trainControl(method = "cv", number = 10, allowParallel = TRUE, verboseIter = TRUE, savePredictions =
"final")

##############################################################################
#  Call ML models for training
#  Run a Random Forest classification model
RF_model<- train(x = samples[-nIndexClass], y = as.factor(samples$Class),method = "rf", ntree = 100, tuneLength
= 10, preProcess = c("center", "scale"), trControl = trainctrl, metric="Kappa")

#  Run a Support Vector Machine (SVM) model
SVM_model<- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "svmRadial", tuneLength =
10, preProcess = c("center", "scale"),trControl = trainctrl, metric="Kappa")

#  Run a k-Nearest Neighbor (kNN) Model
kNN_model <- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "knn",preProcess =
c("center", "scale"),tuneLength = 10, trControl = trainctrl, metric="Kappa")

#  Run an Artificial Neural Network (ANN) model
```

```
ANN_model <- train(x = samples[-nIndexClass], y = as.factor(samples$Class), method = "nnet", preProcess =
c("center", "scale"),tuneLength = 5, trControl = trainctrl, metric="Kappa")

#############################################################################
#Generate error matrix
RF_errorM<- confusionMatrix(RF_model$pred$obs,RF_model$pred$pred)    # RF Error matrix
SVM_errorM <-confusionMatrix(SVM_model$pred$obs,SVM_model$pred$pred) # SVM Error matrix
kNN_errorM <-confusionMatrix(kNN_model$pred$obs,kNN_model$pred$pred) # kNN Error matrix
ANN_errorM <-confusionMatrix(ANN_model$pred$obs,ANN_model$pred$pred) # ANN Error matrix
#############################################################################
# Print/Export the error matrix to the Console
# You can copy and paste the result to a new text file and save it (e.g., RF_ErrorMatrix.txt)
RF_errorM
SVM_errorM
kNN_errorM
ANN_errorM
#############################################################################
# Generate data frame for ensemble analysis of different classifiers
# Note that each classifier randomly split the data into 10 folds
# Thus we need to order them so that they can be matched based on rowIndex
RF<- RF_model$pred[order(RF_model$pred$rowIndex),]
class_RF <- as.numeric((matrix(RF$pred)))

SVM<- SVM_model$pred[order(SVM_model$pred$rowIndex),]
class_SVM <-as.numeric(matrix(SVM$pred))

kNN<- kNN_model$pred[order(kNN_model$pred$rowIndex),]
class_kNN <- as.numeric(matrix(kNN$pred))

ANN<- ANN_model$pred[order(ANN_model$pred$rowIndex),]
class_ANN <- as.numeric(matrix(ANN$pred))

class_actual <- as.numeric(matrix(RF$obs))
#############################################################################
# Conduct McNemar Test
# Define the McNemar function
# Added by Caiyun Zhang
McNemar_Test<-function(Actual, Class1, Class2, nTotal) {

 f11<-0
 f12<-0
 f21<-0
 f22<-0
 z_Value<-0
 for (i in 1:nTotal) {
  if (Actual[i]==Class1[i] && Actual==Class2[i]) {
   f11<-f11+1
  }
  if (Actual[i]==Class1[i] && Actual[i]!=Class2[i]) {
   f12<-f12+1
  }
  if (Actual[i]!=Class1[i] && Actual[i]==Class2[i]) {
   f21<-f21+1
  }
  if (Actual[i]!=Class1[i] && Actual[i]!=Class2[i]) {
   f22<-f22+1
```

```
  }
 }
 z_Value<-abs((f12-f21)/sqrt(f12+f21))
 return(round(z_Value, digits=2))
}

# Call the McNemar function
nTotalRecord<-nrow(samples)
zValue_ANN_SVM<-McNemar_Test(class_actual,class_ANN,class_SVM,nTotalRecord)
zValue_ANN_kNN<-McNemar_Test(class_actual,class_ANN,class_kNN,nTotalRecord)
zValue_ANN_RF<-McNemar_Test(class_actual,class_ANN,class_RF,nTotalRecord)
zValue_SVM_kNN<-McNemar_Test(class_actual,class_SVM,class_kNN,nTotalRecord)
zValue_SVM_RF<-McNemar_Test(class_actual,class_SVM,class_RF,nTotalRecord)
zValue_RF_kNN<-McNemar_Test(class_actual,class_RF,class_kNN,nTotalRecord)
# write out the McNemar test results in csv file
tests<-c("ANN/SVM", "ANN/kNN", "ANN/RF", "SVM/kNN", "SVM/RF", "RF/kNN")
zValues<-c(zValue_ANN_SVM,
zValue_ANN_kNN,zValue_ANN_RF,zValue_SVM_kNN,zValue_SVM_RF,zValue_RF_kNN)
finalOut<-cbind(tests, zValues)
write.csv(finalOut,"McNemarTest.csv")
##############################################################################
# Conduct ensemble analysis classification and error matrix
# Write out error matrix of RF, SVM, ANN and kNN classifiers
# We only combine three classifications from ANN, SVM, and RF
# Added by Caiyun Zhang
nMaxClass<-max(samples$Class) # get the maximum of class number code
nMinClass<-min(samples$Class) # get the minimum of class number code
RF_UA<-0  # define User's accuracy of RF
SVM_UA<-0 # define User's accuracy of SVM
ANN_UA<-0 # define User's accuracy of ANN
kNN_UA<-0 # define User's accuracy of kNN

RF_PA<-0  # define producer's accuracy of RF
SVM_PA<-0 # define producer's accuracy of SVM
ANN_PA<-0 # define producer's accuracy of ANN
kNN_PA<-0 # define producer's accuracy of kNN

# Get the error matrix table
cm_RF <- as.table(RF_errorM)
cm_SVM <- as.table(SVM_errorM)
cm_kNN <- as.table(kNN_errorM)
cm_ANN <- as.table(ANN_errorM)
# Calculate user's accuracy, and producer's accuracy, row total, and column total
ColSum_RF<-rowSums(cm_RF)
ColSum_SVM<-rowSums(cm_SVM)
ColSum_ANN<-rowSums(cm_ANN)
ColSum_kNN<-rowSums(cm_kNN)

RowSum_RF<-colSums(cm_RF)
RowSum_SVM<-colSums(cm_SVM)
RowSum_ANN<-colSums(cm_ANN)
RowSum_kNN<-colSums(cm_kNN)

RF_PA_UA<-RF_errorM$byClass
SVM_PA_UA<-SVM_errorM$byClass
ANN_PA_UA<-ANN_errorM$byClass
```

```
kNN_PA_UA<-kNN_errorM$byClass

for (i in nMinClass:nMaxClass) {

  RF_UA[i]<-RF_PA_UA[i,3] # the third column is UA
  SVM_UA[i]<-SVM_PA_UA[i,3]
  ANN_UA[i]<-ANN_PA_UA[i,3]
  kNN_UA[i]<-kNN_PA_UA[i,3]

  RF_PA[i]<-RF_PA_UA[i,1] # the first column is PA
  SVM_PA[i]<-SVM_PA_UA[i,1]
  ANN_PA[i]<-ANN_PA_UA[i,1]
  kNN_PA[i]<-kNN_PA_UA[i,1]

}

nTotalRowReference<-nrow(samples)
nTotalRowReference
class_EA<-0 # define ensemble analysis prediction
nMajorityVote<-0 # define the majority vote variable
UA_Max<-0 # define the maximum user's accuracy among three votes

for (i in 1:nTotalRowReference) {
  x<-c(class_RF[i], class_SVM[i], class_ANN[i])
  nMajorityVote<-majorityVote(x)
  UA_Max<-0
  # if a majority vote is achieved
  if (max(nMajorityVote$table)!=1)
  {
    class_EA[i]<-nMajorityVote$majority

  }
  # if no agreement is achieved
  else {
    if (RF_UA[class_RF[i]]>UA_Max) {
      class_EA[i]<-class_RF[i]
      UA_Max<-RF_UA[class_RF[i]]
    }
    if (SVM_UA[class_SVM[i]]>UA_Max) {
      class_EA[i]<-class_SVM[i]
      UA_Max<-SVM_UA[class_SVM[i]]
    }
    if (ANN_UA[class_ANN[i]]>UA_Max) {
      class_EA[i]<-class_ANN[i]
    }
    # to test which record has different votes from three classifiers
    print (i)
  }
}

# Print out all training sample prediction  (optional)
 all_pred <- cbind(class_actual,class_RF, class_SVM, class_ANN, class_kNN, class_EA)
 write.csv(all_pred, "all_pred.csv", row.names = FALSE)
# Now read in the prediction. This is weird but this can order the EA error matrix level in the same order as other
classifiers
 predModel <- read.csv("all_pred.csv")
```

```
EA_errorM<- confusionMatrix(as.factor(predModel$class_actual),as.factor(predModel$class_EA))
EA_errorM

# calculate UA and PA of ensemble analysis, and all classifiers out as a csv file
cm_EA <- as.table(EA_errorM)
ColSum_EA<-rowSums(cm_EA)
RowSum_EA<-colSums(cm_EA)
EA_UA<-0
EA_PA<-0
for (i in nMinClass:nMaxClass) {

  EA_UA[i]<-cm_EA[i,i]/ColSum_EA[i]
  EA_PA[i]<-cm_EA[i,i]/RowSum_EA[i]
}

# write out csv error matrix function
WriteErrorM<-function(ETable,ColTotal,UA,RowTotal,PA,OA,Kappa,TotalClass,TotalR,FileName) {

  UA<-round(UA,digits = 2)
  PA<-round(PA,digits = 2)
  RowTotal[TotalClass+1]<-TotalR
  RowTotal[TotalClass+2]<-0
  PA[TotalClass+1]<-0
  PA[TotalClass+2]<-0
  OA[2:TotalClass+2]<-0
  Kappa[2:TotalClass+2]<-0
  AddCol<-cbind(ETable,ColTotal,UA)
  AddRow<-rbind(AddCol,RowTotal,PA, OA, Kappa)
  write.csv(AddRow,FileName)
}

# call the write out error matrix function
A_RF<-RF_errorM$overall # accuracy
OA_RF<-round(A_RF[1],digits = 2) # overall accuracy
Kappa_RF<-round(A_RF[2],digits = 2) # kappa
WriteErrorM(cm_RF,ColSum_RF,RF_UA,RowSum_RF,RF_PA,OA_RF,Kappa_RF, nMaxClass,
nTotalRowReference, "RF_EM.csv")

A_SVM<-SVM_errorM$overall # accuracy
OA_SVM<-round(A_SVM[1],digits = 2) # overall accuracy
Kappa_SVM<-round(A_SVM[2],digits = 2) # kappa
WriteErrorM(cm_SVM,ColSum_SVM,SVM_UA,RowSum_SVM,SVM_PA,OA_SVM,Kappa_SVM, nMaxClass,
nTotalRowReference, "SVM_EM.csv")

A_ANN<-ANN_errorM$overall # accuracy
OA_ANN<-round(A_ANN[1],digits = 2) # overall accuracy
Kappa_ANN<-round(A_ANN[2],digits = 2) # kappa
WriteErrorM(cm_ANN,ColSum_ANN,ANN_UA,RowSum_ANN,ANN_PA,OA_ANN,Kappa_ANN, nMaxClass,
nTotalRowReference, "ANN_EM.csv")

A_kNN<-kNN_errorM$overall # accuracy
OA_kNN<-round(A_kNN[1],digits = 2) # overall accuracy
Kappa_kNN<-round(A_kNN[2],digits = 2) # kappa
WriteErrorM(cm_kNN,ColSum_kNN,kNN_UA,RowSum_kNN,kNN_PA,OA_kNN,Kappa_kNN, nMaxClass,
nTotalRowReference, "kNN_EM.csv")
```

```
A_EA<-EA_errorM$overall # accuracy
OA_EA<-round(A_EA[1],digits = 2) # overall accuracy
Kappa_EA<-round(A_EA[2],digits = 2) # kappa
WriteErrorM(cm_EA,ColSum_EA,EA_UA,RowSum_EA,EA_PA,OA_EA,Kappa_EA, nMaxClass,
nTotalRowReference, "EA_EM.csv")
################################################################################
# Generate final prediction for the input data
# Load in the data for prediction/classification
# Here you need to revise to your own input data for final prediction
# this data set must have same features as the training data and last target column can be set to a default value of 1
# Added and revised by Caiyun Zhang
nRowTotal<-nrow(Data) # get the total number of objects/rows of the input data set
nRowTotal
# Generate prediction from each classifier
pred_RF = predict(RF_model,newdata = Data[-nIndexClass])
pred_SVM = predict(SVM_model,newdata = Data[-nIndexClass])
pred_kNN = predict(kNN_model,newdata = Data[-nIndexClass])
pred_ANN = predict(ANN_model,newdata = Data[-nIndexClass])

# Write out the prediction to csv file
# the output of ID sequence number can be used to join with the FID in ArcGIS shapefile for final mapping
ID<-0:(nRowTotal-1)
# Generate ensemble prediction by combining ANN, SVM, and RF predictions
pred_EA<-0 # define ensemble analysis prediction
nMajorityVote<-0 # define the majority vote variable
UA_Max<-0 # define the maximum user's accuracy among three votes
for (i in 1:nRowTotal) {
 x<-c(pred_RF[i], pred_SVM[i], pred_ANN[i])
 nMajorityVote<-majorityVote(x)
 UA_Max<-0
 # if a majority vote is achieved
 if (max(nMajorityVote$table)!=1)
 {
  pred_EA[i]<-nMajorityVote$majority

 }
 # if no agreement is achieved, three classifiers have completely different prediction
 else {
  if (RF_UA[pred_RF[i]]>UA_Max) {
   pred_EA[i]<-pred_RF[i]
   UA_Max<-RF_UA[pred_RF[i]]
  }
  if (SVM_UA[pred_SVM[i]]>UA_Max) {
   pred_EA[i]<-pred_SVM[i]
   UA_Max<-SVM_UA[pred_SVM[i]]
  }
  if (ANN_UA[pred_ANN[i]]>UA_Max) {
   pred_EA[i]<-pred_ANN[i]
  }
 }
}
EA<-as.numeric(pred_EA)
final_pred <- cbind(ID,pred_RF,pred_SVM,pred_ANN,pred_kNN,EA)
write.csv(final_pred, "FinalPrediction.csv", row.names = FALSE)# Game over
# Game over
################################################
```