Jared Yu

Algorithms for Data Science

Homework 3

Problem 1

Below is a set of three tables where the test statistics are applied to each of the features by iris class. The code is provided at the end as a Code Appendix. (*NOTE:* the trimmed mean, standard deviation, skewness, and kurtosis statistics have been rounded to the 4th decimal place).

Setosa Test Statistics (Trimmed Mean p = 1)

Test Statistics	*Sepal Length*	*Sepal Width*	*Petal Length*	*Petal Width*
Minimum	4.3	2.3	1	0.1
Maximum	5.8	4.4	1.9	0.6
Mean	5.006	3.428	1.462	0.246
Trimmed Mean	5.0042	3.4312	1.4625	0.2417
Standard Deviation	0.3489	0.3753	0.1719	0.1043
Skewness	0.1165	0.0399	0.1032	1.2159
Kurtosis	2.6542	3.7442	3.8046	4.4343

Versicolor Test Statistics (Trimmed Mean p = 1)

Test Statistics	*Sepal Length*	*Sepal Width*	*Petal Length*	*Petal Width*
Minimum	4.9	2	3	1
Maximum	7	3.4	5.1	1.8
Mean	5.936	2.77	4.26	1.326
Trimmed Mean	5.9354	2.7729	4.2687	1.3229
Standard Deviation	0.511	0.3106	0.4652	0.1958
Skewness	0.1022	−0.3519	−0.5882	−0.0302
Kurtosis	2.4012	2.5517	2.9256	2.5122

Virginica Test Statistics (Trimmed Mean p = 1)

Test Statistics	*Sepal Length*	*Sepal Width*	*Petal Length*	*Petal Width*
Minimum	4.9	2.2	4.5	1.4
Maximum	7.9	3.8	6.9	2.5
Mean	6.588	2.974	5.552	2.026
Trimmed Mean	6.5958	2.9729	5.5458	2.0292
Standard Deviation	0.6295	0.3193	0.5463	0.2719
Skewness	0.1144	0.3549	0.5328	−0.1256
Kurtosis	2.9121	3.5198	2.7435	2.3387

From the original analysis of the dataset in the previous programming assignment, it seemed that the last two features, petal length and petal width were best suited to sort the observations into separate categories. That was based on the distribution of the features along a line. In this problem we can look at the features per class in a much more detailed way. Looking at the different features through various test statistics shows that all three iris classes are distinct in their own ways. Each of the four features for the iris classes have both similar and different values for different test statistics. Therefore, it wouldn't make too much sense in this view to say that one iris class is more like another iris class based on just a few values.

Problem 2

Part (a)

The two-class linear discriminant based on Fisher's Linear Discriminant (FLD) was implemented in R. The iris data was first split by the iris species, making the data into three classes of fifty observations each. Class 1 was assigned to the species setosa, Class 2 was assigned to the species versicolor, and Class 3 was assigned to the species virginica. Following the mathematics formula provided in the homework and the Bishop textbook, code was used to generate the optimized weight vector **w** for each pair of classes (i.e., 1 with 2, 1 with 3, and 2 with 3). Also, the decision boundary variable b was similarly created for each pair of classes. Then using the formula (having switched around the values of **x** and **w** for the purposes of matrix multiplication),

$$\underbrace{y}_{1\times1} = \underbrace{\mathbf{x}}_{1\times4}\ \underbrace{\mathbf{w}}_{4\times1} + \underbrace{b}_{1\times1},$$

where **x** is a single observation from one of the classes with four features, a linear discriminant y is created for each observation in the class. If the value of y is greater than or equal to zero, then it is assigned to the second group and the first group otherwise (*NOTE:* If going by the homework formula, the first group is -1, and the second group is +1).

The above process was done twice for each of the three classes. An example is for class 1, it was done once with the **w** and **b** when they were fit to classes 1 and 2, then again when they were fit to classes 1 and 3. Following are the results from each of the pairwise comparisons.

Compare class 1 and class 2

$$\mathbf{X}_{Class\ 1} \times \mathbf{w}_{Class\ 1,Class\ 2} + b_{Class\ 1,Class\ 2}$$

Led to all the observations being correctly assigned to the first class.

$$\mathbf{X}_{Class\ 2} \times \mathbf{w}_{Class\ 1,Class\ 2} + b_{Class\ 1,Class\ 2}$$

Led to all the observations being correctly assigned to the second class.

Compare class 1 and class 3

$$\mathbf{X}_{Class\ 1} \times \mathbf{w}_{Class\ 1,Class\ 3} + b_{Class\ 1,Class\ 3}$$

Led to all the observations being correctly assigned to the first class.

$$\mathbf{X}_{Class\ 3} \times \mathbf{w}_{Class\ 1,Class\ 3} + b_{Class\ 1,Class\ 3}$$

Led to all the observations being correctly assigned to the third class.

Compare class 2 and class 3

$$\mathbf{X}_{Class\ 2} \times \mathbf{w}_{Class\ 2,Class\ 3} + b_{Class\ 2,Class\ 3}$$

Led to 48 observations being correctly assigned to the second class and 2 observations being incorrectly assigned to the third class.

$$\mathbf{X}_{Class\ 3} \times \mathbf{w}_{Class\ 2,Class\ 3} + b_{Class\ 2,Class\ 3}$$

Led to 49 observations being correctly assigned to the third class and 1 observation being incorrectly assigned to the first class.

The results of using LDA on two classes at a time show that LDA has the potential to be an effective tool in classifying observations to the correct group for this dataset. In the case that there is a new set of observations that need labeling, LDA could be useful in identifying the correct iris class that the flower would belong to.

Part (b)

References: https://sthalles.github.io/fisher-linear-discriminant/

References: Applied Multivariate Statistical Analysis, 6th Ed., by Johnson and Wichern

References: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

A similar process to the one above was done when FLD was extended to three classes. This time, however, it was necessary to do different calculations, such as the global mean vector, $\mathbf{m}$, and the between-class scatter matrix, $\mathbf{S}_B$. The process will be shown below:

1) Calculate the global mean vector, $\mathbf{m} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$, where N is the total number of observations in all three classes, and $\mathbf{x}_n$ is a single observation from the dataset. The result is a vector of the four means for all features utilizing all the observations.
2) Calculate the new within-class scatter matrix, $\mathbf{S}_W$, which is the sum of the covariance matrices S_C, from part (a).

3) Calculate the between-class scatter matrix, S_B, which utilizes the following formula:

$$\mathbf{S}_{\mathrm{B}} = \sum_{k=1}^{K} N_k(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T,$$

where K is the total number of classes, N_k is the size of each class k, and $\mathbf{m}_k$ is the mean vector for each class k.

4) Calculate the new weight matrix, $\mathbf{W}$, using the following formula:

$$\mathbf{W} = \max_{D'}\Big(eig(\mathbf{S}_{\mathrm{W}}^{-1}\mathbf{S}_{\mathrm{B}})\Big),$$

where $\mathbf{S}_{\mathrm{W}}^{-1}\mathbf{S}_{\mathrm{B}}$ is the matrix product of the inverse of the within-class scatter matrix and the between-class scatter matrix, $eig()$ is a function that finds the eigenvectors of a matrix, $\max_{D'}()$ is a function that takes the largest D' elements, and D' in this case is set to 3, where $D' > 1$, $D' < D$, and D is the number of dimensions in the dataset (in this case it is four, due to the four features). Therefore, $\mathbf{W}$ is the set of the first three eigenvectors from the product of the inverse of the within-class scatter matrix and the between-class scatter matrix.

5) Perform Fisher's classification procedure based on sample discriminants. The rule for the procedure is as follows (taken from the textbook referenced above):
Allocate $\mathbf{x}$ to class k if

$$\sum_{j=1}^{r}\left[\hat{\mathbf{a}}_j'(\mathbf{x} - \bar{\mathbf{x}}_k)\right]^2 \leq \sum_{j=1}^{r}\left[\hat{\mathbf{a}}_j'(\mathbf{x} - \bar{\mathbf{x}}_i)\right]^2 \text{ for all } i \neq k,$$

where $\hat{\mathbf{a}}_j'$ is the $j'th$ eigenvector from $\mathbf{W}$, and $r = D'$. The above procedure will assign an observation to a class k, if the sum of squares for the scaled difference between the observation and the class mean of every other class is larger or equal. A function called distance_measure() was created to help with this process.

6) Create a confusion matrix to show the results of the classification, shown below.

The following is a confusion matrix of the results from assigning each observation to a class.

	$Class\ 1\ Predicted$	$Class\ 2\ Predicted$	$Class\ 3\ Predicted$
$Class\ 1\ Actual$	50	0	0
$Class\ 2\ Actual$	0	48	2
$Class\ 3\ Actual$	0	1	49

The accuracy is as follows:

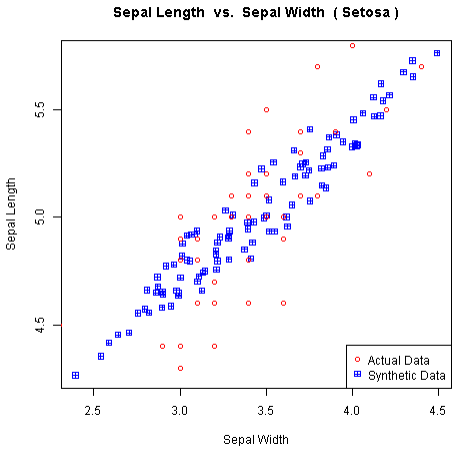
$$Accuracy = \frac{50 + 48 + 49}{50 + 58 + 49 + 1 + 2} = 0.98$$

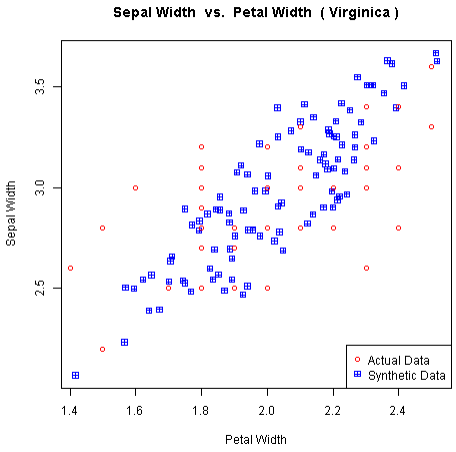
The results of using LDA on all three classes are good. The accuracy of the confusion matrix shows that LDA can be an effective tool when it is extended to multiple classes for this dataset. Once again, like in part (a), if there are a new set of observations that need labeling, LDA could do a good job of assigning the new observations to the correct class.

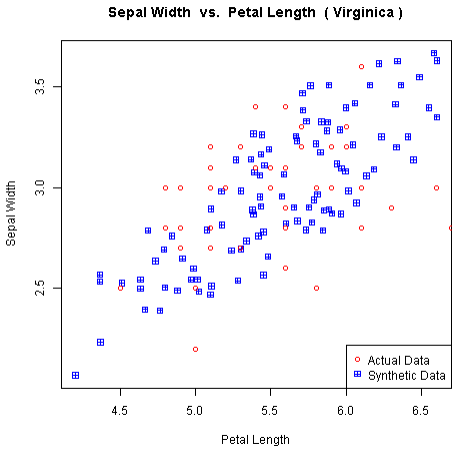
Problem 3

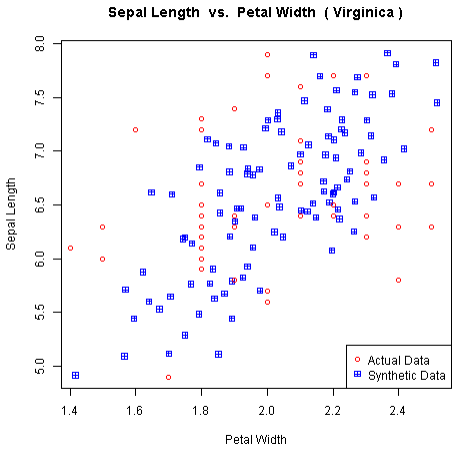
References: https://superuser.com/questions/940771/how-can-i-resize-multiple-images-in-a-ms-word-document

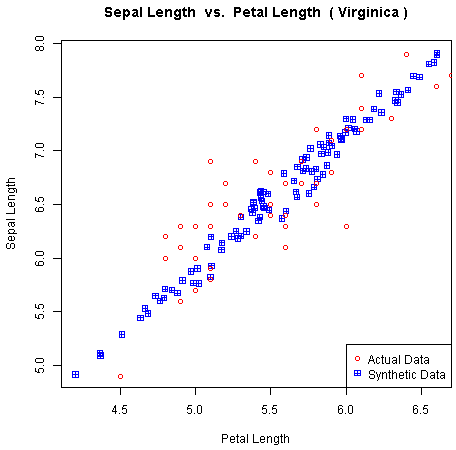
Below are 18 plots showing the 6 pairwise combinations of the 4 features for each of the 3 classes. The red circles indicate the actual data, while the blue squares indicate the synthetic data. The method for creating the synthetic observations come from the Matlab code provided by the professor. The plotting was done in RStudio.

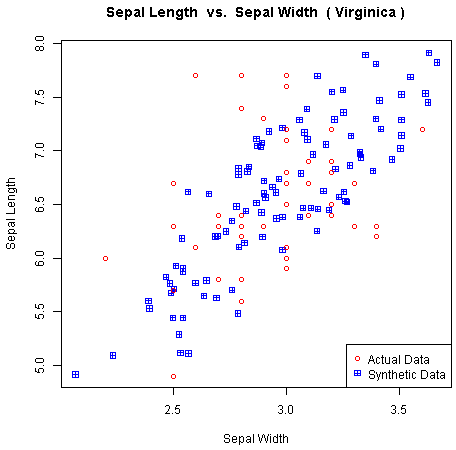


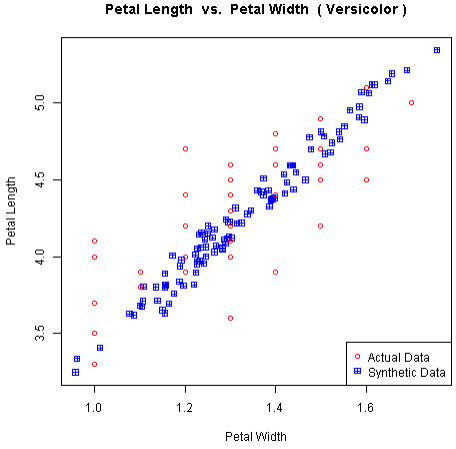


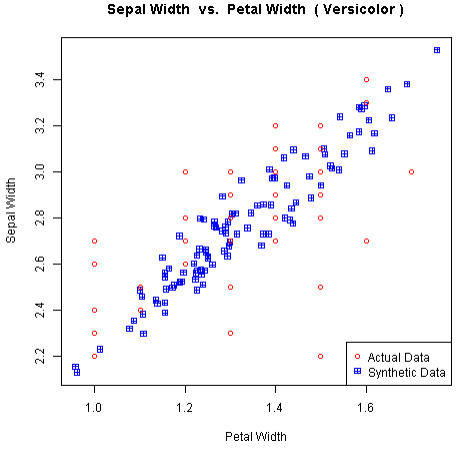


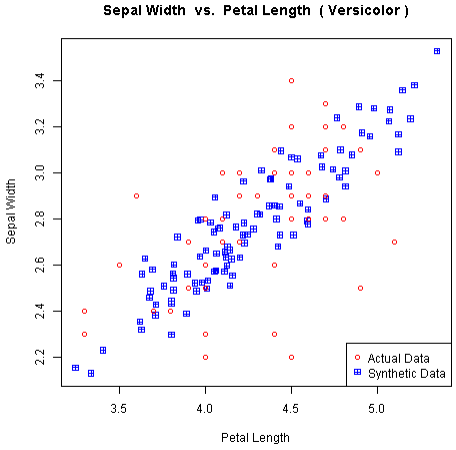


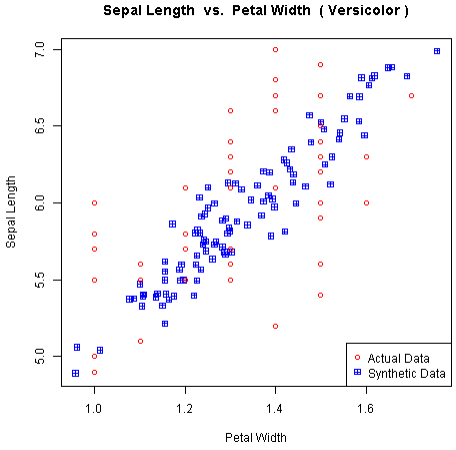

Sepal Length vs. Petal Length (Virginica)
Sepal Length
Petal Length
Actual Data
Synthetic Data

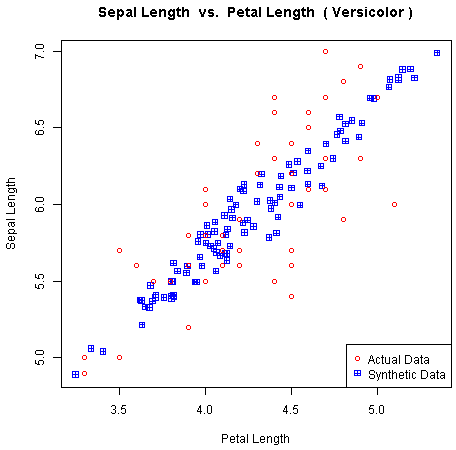

Sepal Length vs. Sepal Width (Virginica)
Sepal Length
Sepal Width
Actual Data
Synthetic Data

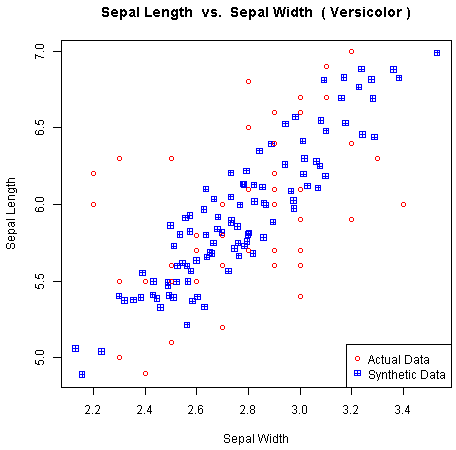

Petal Length vs. Petal Width (Versicolor)
Petal Length
Petal Width
Actual Data
Synthetic Data

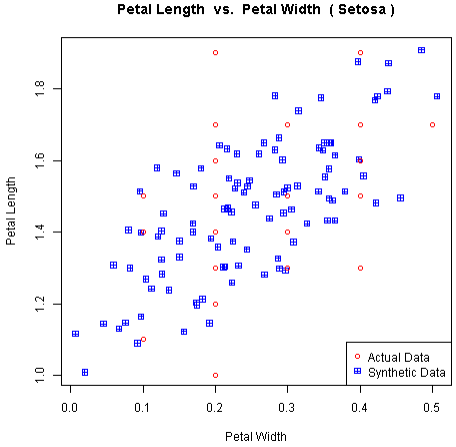

Sepal Width vs. Petal Width (Versicolor)
Sepal Width
Petal Width
Actual Data
Synthetic Data

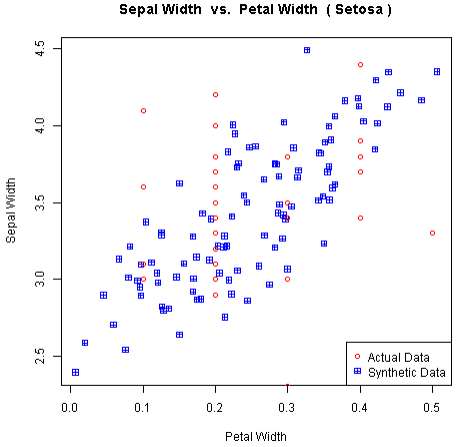

Sepal Width vs. Petal Length (Versicolor)
Sepal Width
Petal Length
Actual Data
Synthetic Data

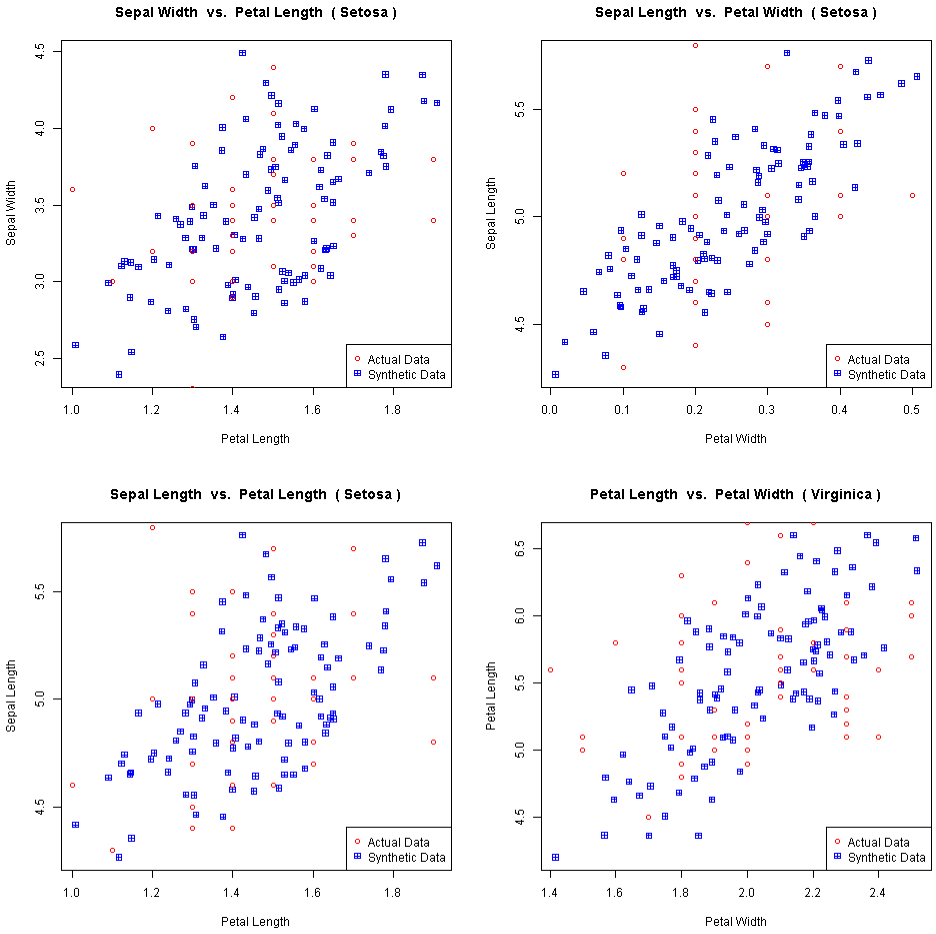

Sepal Length vs. Petal Width (Versicolor)
Sepal Length
Petal Width
Actual Data
Synthetic Data


Sepal Length vs. Petal Length (Versicolor)
Sepal Length
Petal Length
Actual Data
Synthetic Data


Sepal Length vs. Sepal Width (Versicolor)
Sepal Length
Sepal Width
Actual Data
Synthetic Data


Petal Length vs. Petal Width (Setosa)
Petal Length
Petal Width
Actual Data
Synthetic Data


Sepal Width vs. Petal Width (Setosa)
Sepal Width
Petal Width
Actual Data
Synthetic Data



The results of the plotting show that given a set of real data, it is possible to use a random data generating technique like the one shown above to simulate data so that there is a larger dataset to work with. In this case, 150 observations were increased to 450 observations. Having a larger dataset makes it possible for a data scientist using a modeling process to tune a model to certain parameters for the purpose of developing a baseline to work with. Once the algorithm is used and new observations are gathered, it is possible then to fine tune the parameters of the model so that there is greater accuracy in the prediction.

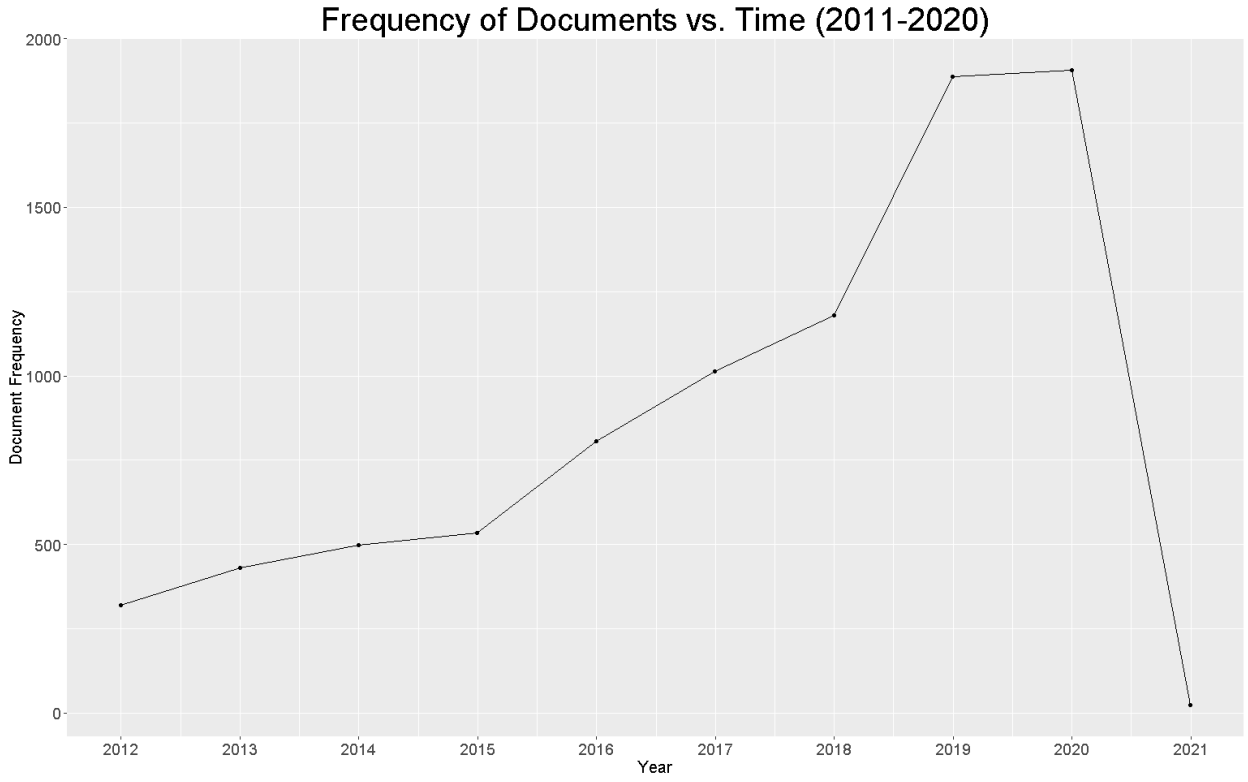
Problem 4

Part (a)

From the plot of the frequency of documents in the past 10 years from Scopus, it is apparent that there has been a dramatic rise in documents related to data science and machine learning. In Scopus, the following was typed in without quotes, “data science AND machine learning.” The frequency in the beginning around 2011 (the axis shows 2012, but the date is specified as the last day of 2011) was quite moderate in comparison, trending upwards slowly until around 2015. From that point on there began a large increase in the rate of documents related to the two

subjects. Then by around 2018, there was a large spike again which has quickly slowed down. It is interesting that there are no more documents for 2019, even though the year has not ended, and documents have already been labeled 2020. This explains why there is a sudden large drop in the frequency of papers. The year 2020 has not yet begun and so it wouldn't make sense to try and estimate the frequency of the documents when the year has not yet even began, based on the plot alone.

It can be gathered that data science and machine learning have grown increasingly in popularity over the past decade. It is likely that due to the increase in people being aware of their existence that more people are therefore generating documents related to the topics. A possible reason is that certain advancements in technology have made it easier for more people to easily access data science and machine learning tools, in addition to them being better able to use these fields effectively.



Part (b)

To determine the author of a paper, it is necessary to export at least the Authors column from each document. To find the institution collaborations, it would be necessary to export the Document Details for each document to find out this information. From the Document Details page, it is possible to see more information about both the authors of the document along with the institutions that they are connected to.

Part (c)

The field that would require data cleansing is the Document Details page. This page is filled with extra information not relevant to understanding who the authors are or the institutions that collaborated on the document. The only section of the Document Details page that gives the relevant information is the section under the title, where the authors are listed first, and then the institutions that collaborated on the document. Much of the other information seems irrelevant towards understanding who the authors are or the institutions who have collaborated. There are two exceptions, the first would be at the end of the Abstract section, where a publisher is included. The second exception may be at the end beneath the keywords, ISSN, DOI, Source Type, etc., where contact for an author is also included.

Code Appendix (the font size as been shrunk to 9 so that the code doesn't run to the next line):

```
# Jared Yu
# Algorithms for Data Science
# Homework 3

# Libraries
library("ggplot2")

### Problem 1
# Load data
iris_data <- iris

# Reference: https://stackoverflow.com/questions/40721960/insertion-sort-algorithm-in-r
# The insertionsort_function is a function that utilizes the
# insertion sort method to sort an array from minimum to maximum.
# The function is borrowed from a stackoverflow post which is linked
# above. This function will be utilized to help calculate the
# minimum and maximum values.
insertionsort_function <- function(array_A){
  # Loop from index 2 to the last element
  for (index_j in 2:length(array_A)) {
    key <- array_A[index_j]
    # Insert array_A[index_j] into sorted sequence array_A[1,...,index_j-1]
    index_i <- index_j - 1
    while (index_i > 0 && array_A[index_i] > key) {
      array_A[(index_i + 1)] <- array_A[index_i]
      index_i <- index_i - 1
    }
    array_A[(index_i + 1)] <- key
  }

  return(array_A)  # Return the sorted array
}

# The test_statistic_minimum function is used to calculate the
# minimum test statistic from a given input vector. It utilizes
# the insertionsort_function to sort the input vector from
# minimum to maximum using the insertion sort method. It then
# takes the first element from this sorted array and returns it.
test_statistic_minimum <- function(input_vector) {
  # Sort the input vector using insertion sort from
  # minimum to maximum, then take the first item
```

```r
  minimum_element <- insertionsort_function(input_vector)[1]

  return(minimum_element)  # Return the minimum
}

# The test_statistic_maximum function is used to calculate the
# maximum test statistic from a given input vector. It utilizes
# the insertionsort_function to sort the input vector from
# minimum to maximum using the insertion sort method. It then
# takes the last element from this sorted array and returns it.
test_statistic_maximum <- function(input_vector) {
  # Sort the input vector using insertion sort from
  # minimum to maximum, then take the last item
  maximum_element <- insertionsort_function(input_vector)[length(input_vector)]

  return(maximum_element)  # Return the maximum
}

# The test_statistic_mean function is used to calculate the
# mean test statistic from a given input vector. It sums
# all of the elements in the input vector together and then
# divides them by the number of elements in the vector. It
# then returns this value.
test_statistic_mean <- function(input_vector) {
  # Calculate the mean by summing the elements of the
  # vector and dividing by the number of elements
  vector_mean <- sum(input_vector) / length(input_vector)

  return(vector_mean)  # Return the mean
}

# The test_statistic_trimmed_mean function calculates the trimmed
# mean from an input vector. It does so by first sorting the
# vector from low to high. Then, it removes a p number of elements
# from the beginning and end of the vector. Then it calculates
# the trimmed mean by summing the trimmed vector and dividing it
# by the numeber of elements minus 2 * p.
test_statistic_trimmed_mean <- function(input_vector, p_elements) {
  # Sort the input vector from low to high
  sorted_vector <- insertionsort_function(input_vector)
  # Trim the vector, removing p elements from the beginning and end
  trimmed_vector <- sorted_vector[(1 +
                    p_elements):(length(input_vector) - p_elements)]
  # Calculate the trimmed mean, summing the trimmed vector and
  # dividing by the length of the vector minus 2 * p
  trimmed_mean <- sum(trimmed_vector) / (length(input_vector) - 2 * p_elements)

  return(trimmed_mean)  # Return the trimmed mean
}

# The test_statistic_standard_deviation function calculates the
# standard deviation from an input vector. It does so by first
# calculating the mean using the test_statistic_mean() function.
# Then it subtracts this mean from each of the elements in the
# vector and squares the value. This squared difference is then
# summed throughout the vector. The sum of the squared difference
# is then divided by the number of elements in the vector and
# the square root is taken. The standard deviation is then returned.
test_statistic_standard_deviation <- function(input_vector) {
  # Calculate the mean (mu) of the input vector
  mu_vector <- test_statistic_mean(input_vector = input_vector)
```

```r
  # Calculate the sum of the squared difference between each
  # element in the vector and the mu
  summed_squared_difference <- sum((input_vector - mu_vector)^2)

  # Calculate the standard deviation by taking the square root
  # of the summed squared difference divided by the number of elements
  standard_deviation <- sqrt(summed_squared_difference / length(input_vector))

  return(standard_deviation)  # Return the standard deviation
}

# The test_statistic_skewness function calculates the skewness of
# an input vector. It does so by first calculating the mean (mu)
# using the test_statistic_mean() function. It then takes the
# summed cubed difference between the elements in the vector
# and the mean of the vector. Then it calculates the skewness
# of the vector by dividing the summed cubed difference by the
# number of elements in the vector and then dividing that value
# by the cube of the standard deviation. It then returns the skewness.
test_statistic_skewness <- function(input_vector) {
  # Calculate the mean (mu) of the input vector
  mu_vector <- test_statistic_mean(input_vector = input_vector)

  # Calculate the sum of the cubed difference between each
  # element in the vector and the mu
  summed_cubed_difference <- sum((input_vector - mu_vector)^3)

  # Calculate the skewness, first dividing the summed cubed difference
  # by the number of elements in the input vector. Then dividng this
  # value by the cubed standard deviation of the input vector.
  vector_skewness <- (summed_cubed_difference / length(input_vector)) /
                      (test_statistic_standard_deviation(input_vector =
                                                         input_vector)^3)

  return(vector_skewness)  # Return the skewness
}

# The test_statistic_kurtosis function calculates the kurtosis of
# an input vector. It does so by first calculating the mean (mu)
# using the test_statistic_mean() function. It then takes the
# summed quartic difference between the elements in the vector
# and the mean of the vector. Then it calculates the kurtosis
# of the vector by dividing the summed quartic difference by the
# number of elements in the vector and then dividing that value
# by the fourth power of the standard deviation. It then returns
# the kurtosis.
test_statistic_kurtosis <- function(input_vector) {
  # Calculate the mean (mu) of the input vector
  mu_vector <- test_statistic_mean(input_vector = input_vector)

  # Calculate the sum of the quartic difference between each
  # element in the vector and the mu
  summed_quartic_difference <- sum((input_vector - mu_vector)^4)

  # Calculate the kurtosis, first dividing the summed quartic difference
  # by the number of elements in the input vector. Then dividng this
  # value by the quartic standard deviation of the input vector.
  vector_kurtosis <- (summed_quartic_difference / length(input_vector)) /
                      (test_statistic_standard_deviation(input_vector =
                                                         input_vector)^4)
```

```r
  return(vector_kurtosis)  # Return the skewness
}

# Reference: https://stackoverflow.com/questions/7381455/filtering-a-data-frame-by-values-in-a
-column
# Subset the iris data by each of the classes
iris_setosa <- iris_data[iris_data$Species == 'setosa', -ncol(iris_data)]
iris_versicolor <- iris_data[iris_data$Species == 'versicolor', -ncol(iris_data)]
iris_virginica <- iris_data[iris_data$Species == 'virginica', -ncol(iris_data)]

# References: https://www.r-bloggers.com/applying-multiple-functions-to-data-frame/
# The test_statistics function is used as a modified version of a function from
# the link above. The r-bloggers page provides a function which can apply
# multiple functions to a dataframe which is done by cominbing the test_statistics
# function with an sapply() command. The test_statistics applies the various
# test statistic functions which are created above. The test_staitstics function
# includes the minimum, maximum, mean, trimmed mean, standard deviation,
# skewness, and kurtosis functions which are applied to each column of a
# dataframe using sapply().
test_statistics <- function(data_frame, p = 1) {
  # Combine the test statistics of minimum, maximum, mean, trimmed mean,
  # standard deviation, skewness, and kurtosis into a vector.
  data_test_statistics <- c(F_min <- test_statistic_minimum(data_frame),
    F_max <- test_statistic_maximum(data_frame),
    F_mean <- test_statistic_mean(data_frame),
    F_trimmed_mean <- round(test_statistic_trimmed_mean(data_frame, p_elements = p), 4),
    F_standard_deviation <- round(test_statistic_standard_deviation(data_frame), 4),
    F_skewness <- round(test_statistic_skewness(data_frame), 4),
    F_kurtosis <- round(test_statistic_kurtosis(data_frame), 4))

  # Return the vector of test statistics (used within sapply)
  return(data_test_statistics)
}

# The test_statistics_table is a function which will generate the test
# statistics of minimum, maximum, mean, trimmed mean, standard deviation
# skewness, and kurtosis of an iris class. It will output a table of the
# test statistics for a class from the iris dataset. It will describe
# which iris class is being shown, along with the p number argument for
# the test statistic trimmed mean.
test_statistics_table <- function(iris_class, class_name, p_number = 1) {
  # Calculate the test statistics dataframe using sapply() with the
  # function test_statistics().
  test_statistics_dataframe <- sapply(iris_class, function(index)
    test_statistics(data_frame = index, p = p_number))

  # Set the row names for the dataframe to show the corresponding test statistics
  row.names(test_statistics_dataframe) <- c("Minimum", "Maximum", "Mean",
                                    "Trimmed Mean", "Standard Deviation",
                                    "Skewness", "Kurtosis")

  # Print the iris class along with the p number of the trimmed mean
  print(paste(c(class_name, " Test Statistics", " (Trimmed Mean p = ",
                p_number, ")"), collapse = ""))

  # Return the table
  return(test_statistics_dataframe)
}

# References: https://sejdemyr.github.io/r-tutorials/basics/tables-in-r/
```

```r
# Show the test statistics for the different iris classes
write.table(test_statistics_table(iris_class = iris_setosa, class_name = "Setosa"),
            file = "table1.txt", sep = ",", quote = FALSE, row.names = T)
write.table(test_statistics_table(iris_class = iris_versicolor, class_name = "Versicolor"),
            file = "table2.txt", sep = ",", quote = FALSE, row.names = T)
write.table(test_statistics_table(iris_class = iris_virginica, class_name = "Virginica"),
            file = "table3.txt", sep = ",", quote = FALSE, row.names = T)

### Problem 2
### Part (a)
# List version
iris_list <- split(iris_data[,1:4], f = iris_data$Species)

# List of class mean vectors
mean_list <- lapply(iris_list, function(index) apply(index, 2, mean))

# List of covariance matrices for classes 1, 2, and 3
covariance_list <- lapply(iris_list, function(index) 49*cov(index))

# Calculate within-class scatter matrices
class_size_k_C <- nrow(iris_list[[1]])
total_samples_k <- nrow(rbind(as.data.frame(iris_list[[1]]),
                              as.data.frame(iris_list[[2]])))
a_priori <- class_size_k_C / total_samples_k

# Create the inverse within class scatter matrix 'S_W^-1'
scatter_matrix_within_class_1_2_inverse <- solve((covariance_list[[1]] +
                                                  covariance_list[[2]])*a_priori)
scatter_matrix_within_class_1_3_inverse <- solve((covariance_list[[1]] +
                                                  covariance_list[[3]])*a_priori)
scatter_matrix_within_class_2_3_inverse <- solve((covariance_list[[2]] +
                                                  covariance_list[[3]])*a_priori)

# Create the weight vector 'w'
weight_vector_1_2 <- scatter_matrix_within_class_1_2_inverse %*%
  (mean_list$versicolor - mean_list$setosa)
weight_vector_1_3 <- scatter_matrix_within_class_1_3_inverse %*%
  (mean_list$virginica - mean_list$setosa)
weight_vector_2_3 <- scatter_matrix_within_class_2_3_inverse %*%
  (mean_list$virginica - mean_list$versicolor)

# Create the decision boundary variable 'b'
decision_boundary_1_2 <- -0.5 * (t(weight_vector_1_2) %*%
                                   (mean_list$setosa + mean_list$versicolor))
decision_boundary_1_3 <- -0.5 * (t(weight_vector_1_3) %*%
                                   (mean_list$setosa + mean_list$virginica))
decision_boundary_2_3 <- -0.5 * (t(weight_vector_2_3) %*%
                                   (mean_list$versicolor + mean_list$virginica))

# Compare class 1 versus class 2
y_1_2_1 <- as.matrix(iris_list$setosa) %*% weight_vector_1_2 +
  as.numeric(decision_boundary_1_2)
y_1_2_2 <- as.matrix(iris_list$versicolor) %*% weight_vector_1_2 +
  as.numeric(decision_boundary_1_2)
table(ifelse(y_1_2_1 >= 0, 2, 1))
table(ifelse(y_1_2_2 >= 0, 2, 1))

# Compare class 1 versus class 3
y_1_3_1 <- as.matrix(iris_list$setosa) %*% weight_vector_1_3 +
  as.numeric(decision_boundary_1_3)
y_1_3_3 <- as.matrix(iris_list$virginica) %*% weight_vector_1_3 +
```

```r
  as.numeric(decision_boundary_1_3)
table(ifelse(y_1_3_1 >= 0, 3, 1))
table(ifelse(y_1_3_3 >= 0, 3, 1))

# Compare class 2 versus class 3
y_2_3_2 <- as.matrix(iris_list$versicolor) %*% weight_vector_2_3 +
  as.numeric(decision_boundary_2_3)
y_2_3_3 <- as.matrix(iris_list$virginica) %*% weight_vector_2_3 +
  as.numeric(decision_boundary_2_3)
table(ifelse(y_2_3_2 >= 0, 3, 2))
table(ifelse(y_2_3_3 >= 0, 3, 2))

### Part (b)
# Global dataframe
class_1_2_3 <- iris_data[, -ncol(iris_data)]

# Global mean vector
n_size_global <- nrow(iris_data)
one_vector_global <- rep(1, n_size_global)
global_mean_vector <- (1 / n_size_global) *
  t(class_1_2_3) %*% one_vector_global

# Reference: https://stackoverflow.com/questions/33899698/r-sum-element-x-in-list-of-vector?rq
=1
# Within-class scatter matrix
scatter_matrix_within_class <- Reduce("+", covariance_list)

# Between-class scatter matrix
class_size_n <- nrow(iris_list[[1]])
scatter_matrix_between_class <- Reduce("+",
  lapply(mean_list, function(index) (index - global_mean_vector) %*%
  t(index - global_mean_vector))) * class_size_n

# Inverse within-class scatter matrix
inverse_scatter_matrix_within_class <- solve(scatter_matrix_within_class)

# Weight matrix for all 3 classes
weight_matrix_1_2_3 <- eigen(inverse_scatter_matrix_within_class %*%
                             scatter_matrix_between_class)$vectors[, 1:3]

# The distance_measure function is used to help calculate the Fisher's Classification
# Procedure Based on Sample Discriminants, from  p.631 in Applied Multivariate Statistical
# Analysis, 6th Ed. by Johnson and Wichern.
distance_measure <- function(data_set = class_1_2_3, class_mean = mean_class_1,
                             eigenvector_matrix = weight_matrix_1_2_3, dimension_reduction =
3) {
  # Create an nx1 dimensional vector of ones
  one_vector <- rep(1, nrow(data_set))

  # Calculate the scaled difference of an observation from each of the class means
  observation_mean_deviation <- as.matrix(data_set - (one_vector %*% t(class_mean))) %*%
    eigenvector_matrix[,1:dimension_reduction]

  # Return the sum of squares for the scaled deviations
  return(apply(observation_mean_deviation^2, 1, sum))
}

# Matrix for the scaled deviations of each class from each of the class means
distance_measurement_matrix <- sapply(mean_list,
              function(index) distance_measure(class_mean = index))
```

```r
# Create a confusion matrix for the Fisher classification
predicted_label <- apply(distance_measurement_matrix, 1, which.min)
true_label <- rep(1:3, each = 50)
table(true_label, predicted_label)

### Problem 3
# Create a list for the minimum and maximum values of each class
iris_minimum_list <- lapply(iris_list, function(x) apply(x, 2, min))
iris_maximum_list <- lapply(iris_list, function(x) apply(x, 2, max))

set.seed(100)  # Set seed so that random numbers are constant
# The synthesize_iris_data function is a function to be used on each class of the
# iris data. It will generate 100 new observations for each class that are random.
# It will rotate them using a covariance matrix and normalize them with the assistance
# of a helper function, normalize_column().
synthesize_iris_data <- function(list_element_number = 1, iris_list_data = iris_list,
                                 covariance_list_data = covariance_list,
                                 minimum_list = iris_minimum_list,
                                 maximum_list = iris_maximum_list) {
  # Random matrix of 100 observations
  random_matrix <- matrix(rep(runif(n = 400)), ncol = 4)

  # Rotate the random matrix according to the covariance matrix
  rotated_data <- as.data.frame(random_matrix %*%
                                  covariance_list_data[[list_element_number]])

  # Normalize each column of the rotated data
  # trying to use apply on the rotated data, 4 columns, output a vector for each column?
  normalized_data <- sapply(seq_along(rotated_data), function(index)
    normalize_column(list_element = index, random_data = rotated_data,
                     iris_class = list_element_number))

  # Return the normalized data
  return(normalized_data)
}

# The normalize_column function is a helper function that is used within
# the synthesize_iris_data() function. It is repeated for each column
# of a given class, where it will normalize the random data.
normalize_column <- function(list_element, random_data, iris_class,
                             minimum_list = iris_minimum_list,
                             maximum_list = iris_maximum_list) {
  # Select the current column from the rotated data
  rotated_data_column <- random_data[, list_element]

  # Create a Pmin and Pmax for the min and max of the rotated data
  random_min <- min(rotated_data_column)
  random_max <- max(rotated_data_column)

  # Create a min and max from the actual iris data
  actual_min <- iris_minimum_list[[iris_class]][list_element]
  actual_max <- iris_maximum_list[[iris_class]][list_element]

  # Normalize the rotated data
  rotated_data_column <- ((rotated_data_column - random_min) /
        (random_max - random_min)) * (actual_max - actual_min) + actual_min

  # Return the normalized data
  return(rotated_data_column)
}
```

```r
# Reference: https://stackoverflow.com/questions/9950144/access-lapply-index-names-inside-fun
# Generate 100 observations per class, rotated and normalized
synthetic_data_list <- lapply(seq_along(iris_list), function(index)
  synthesize_iris_data(list_element_number = index))

# Mean shift the data
synthetic_mean <- lapply(synthetic_data_list, function(index) apply(index, 2, mean))
synthetic_difference_list <- lapply(seq_along(synthetic_mean), function(index)
  synthetic_mean[[index]] - mean_list[[index]])

one_vector_synthetic <- rep(1, 100)  # Use ones vector to generate mean vector

synthetic_mean_100 <- lapply(seq_along(synthetic_difference_list), function(index_i)
  sapply(seq_along(iris_list[[1]]), function(index_j)
  as.matrix(one_vector_synthetic) %*% synthetic_difference_list[[index_i]][index_j]))

synthetic_iris_list <- lapply(seq_along(synthetic_data_list), function(index)
  synthetic_data_list[[index]] - synthetic_mean_100[[index]])

# Create all plot combinations for each class
# Reference: https://stackoverflow.com/questions/17171148/non-redundant-version-of-expand-grid
# Reference: https://stackoverflow.com/questions/18509527/first-letter-to-upper-case/18509816
feature_names <- c("Sepal Length", "Sepal Width",  # Feature names for plot labels
                   "Petal Length", "Petal Width")

# The synthetic_plots function is used to plot all of the combinations for each
# of the iris classes. There are a total of 18 plots, where each class needs
# to create 6 plots for each pairwise combination of the 4 features. The function
# will create the plots and save a '.png' to a local folder designated in the
# folder_path argument. The plots show the actual 2 features of the iris data
# in addition to plotting the synthetic iris data. The actual data are shown
# as red circles, the synthetic data are represented with blue squares.
synthetic_plots <- function(folder_path = "./problem_3_plots/",
                            class_index = 1, iris_list_data = iris_list,
                            synthetic_iris_list_data = synthetic_iris_list,
                            column_names = feature_names) {
  # Enumerate all possible combinations of 4 features
  plot_combinations <- t(combn(x = 1:4, m = 2))

  # Generate a plot for each combination for the given class
  apply(plot_combinations, 1, function(index) {

    # Save to file
    png(paste(folder_path, "class_", class_index,  "_",
            "feature_", index[1], "_", index[2], ".png", sep = ''))

    # Plot synthetic data
    plot(synthetic_iris_list_data[[class_index]][,index[2]],
         synthetic_iris_list_data[[class_index]][,index[1]],
         main = paste(column_names[index[1]], " vs. ", column_names[index[2]],
                      " (", paste0(toupper(substr(names(iris_list_data)[class_index], 1, 1)),
                      substr(names(iris_list_data)[class_index], 2,
                      nchar(names(iris_list_data)[class_index]))), ")"),
         xlab = column_names[index[2]], ylab = column_names[index[1]],
         col = "blue", pch = 12)

    # Plot actual data
    points(iris_list_data[[class_index]][,index[2]],
           iris_list_data[[class_index]][,index[1]], col = 'red')

    # Legend
```

```
    legend("bottomright", legend = c("Actual Data", "Synthetic Data"),
           col = c("red", "blue"), pch = c(1, 12))

    dev.off()  # Close plot
  })
}

# Create scatterplot for all combinations
sapply(1:3, function(index)
  synthetic_plots(class_index = index))

### Problem 4
# Reference: https://stackoverflow.com/questions/41815365/error-invalid-input-date-trans-works
-with-objects-of-class-date-only
# Reference: https://www.statmethods.net/input/dates.html
# Create the X-axis for each of the dates in the graph (up until the present)
year_x_axis <- as.Date(c("2011-12-31", "2012-12-31", "2013-12-31", "2014-12-31",
"2015-12-31", "2016-12-31", "2017-12-31", "2018-12-31", "2019-12-31", "2020-12-31"))

# Create the Y-axis consisting of the frequency for each of the documents
frequency_y_axis <- c(321, 431, 497, 535, 806, 1014, 1179, 1887, 1906, 24)

# Create the dataframe of the scoupus data using the X and Y axes
scopus_data <- data.frame(Year = year_x_axis, Frequency = frequency_y_axis)

# Plot the information using ggplot
ggplot(scopus_data, aes(x = year_x_axis, y = frequency_y_axis)) +
  geom_line() + ggtitle(label = "Frequency of Documents vs. Time (2011-2020)") +
  theme(plot.title = element_text(size = 30, hjust = 0.5),
        axis.title.x = element_text(size = 15),
        axis.title.y = element_text(size = 15)) +
  xlab("Year") + ylab("Document Frequency") +
  geom_point(mapping = aes(x = year_x_axis, y = frequency_y_axis)) +
  scale_x_date(date_breaks = "1 year", date_labels = "%Y")
```