Classifying an unknown Iris flowers based on four measurements

Introduction

In this course, a lot of the high level problems we are asked to solve boil down to ranking something. I'm interested in machine learning, so today I want to explore another common problem in machine learning and data science using the skills I've learned in this course: categorizing something.

The data that we are going to be using is a very famous dataset that comes from Fisher's paper about classifying three types of irises based on 4 characteristics: sepal length, sepal width, petal length, and petal width. In other words, this is a pattern recognition problem.

```
#download an image of an iris (optional)
iris_pic_url <- "https://www.fs.fed.us/wildflowers/beauty/iris/images/blueflagiris_flower_lg.jpg"
download.file(iris_pic_url, 'iris.jpg')
```

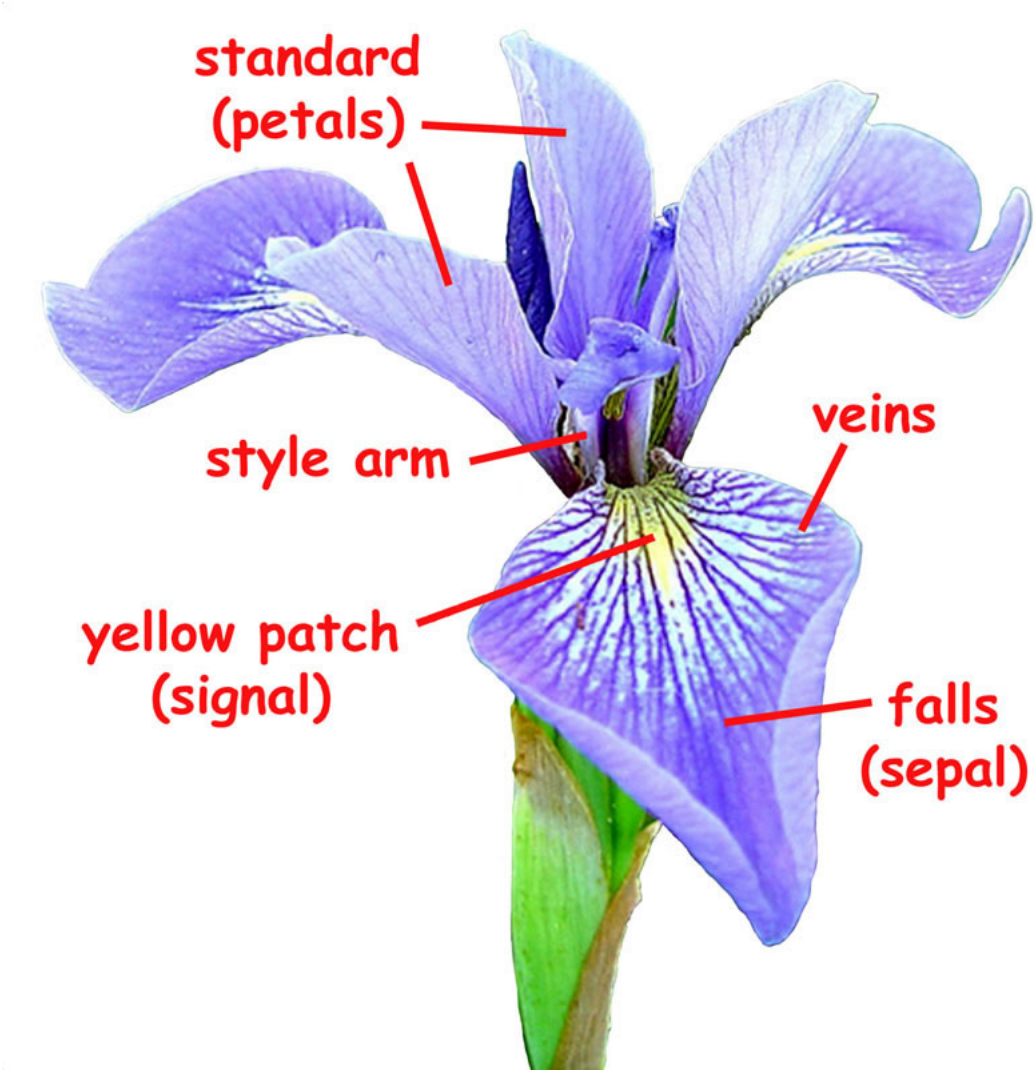


Diagram of an iris

Getting familiar with the data

- Loading needed libraries.

```
library(ggplot2)
library(readr)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

- Download the data.

```
#downloading the data
iris_url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/bezdekIris.data"
download.file(iris_url, 'iris_data.csv')

#downloading the data dictionary
iris_dictionary_url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.names"
download.file(iris_dictionary_url, 'iris_data_dictionary.txt')

#read in the data
column_names = c("sepal_length", "sepal_width", "petal_length", "petal_width",
                 "iris_class")
iris_dat <- read_csv("iris_data.csv", col_names = column_names)
```

```
## Parsed with column specification:
## cols(
##   sepal_length = col_double(),
##   sepal_width = col_double(),
##   petal_length = col_double(),
##   petal_width = col_double(),
##   iris_class = col_character()
## )
```

- Quick glance at the data itself.

```
summary(iris_dat)
```

```
##   sepal_length    sepal_width     petal_length    petal_width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##   iris_class
##  Length:150
##  Class :character
##  Mode  :character
##
##
##
```

Problem: Classifying an unknown Iris into one of the three types

Let's say you are given the measurements of an unknown iris and asked to classify it.

```
#simulate an unknown iris
#set the seed
set.seed(1)

#creating the unknown iris measurements
unknown_iris = c()
for (meas in c(1:4)) {
  unknown_iris = c(unknown_iris, runif(1,
                                       min=min(iris_dat[,meas]),
                                       max=max(iris_dat[,meas])))
}

#you should get these numbers
print(unknown_iris)
```

```
## [1] 5.255831 2.893097 4.379835 2.279699
```

```
#creating a data table for the unknown iris
u_iris <- data.frame(sepal_length = unknown_iris[1],
                     sepal_width = unknown_iris[2],
                     petal_length = unknown_iris[3],
                     petal_width = unknown_iris[4],
                     iris_class = "unknown"
)
```

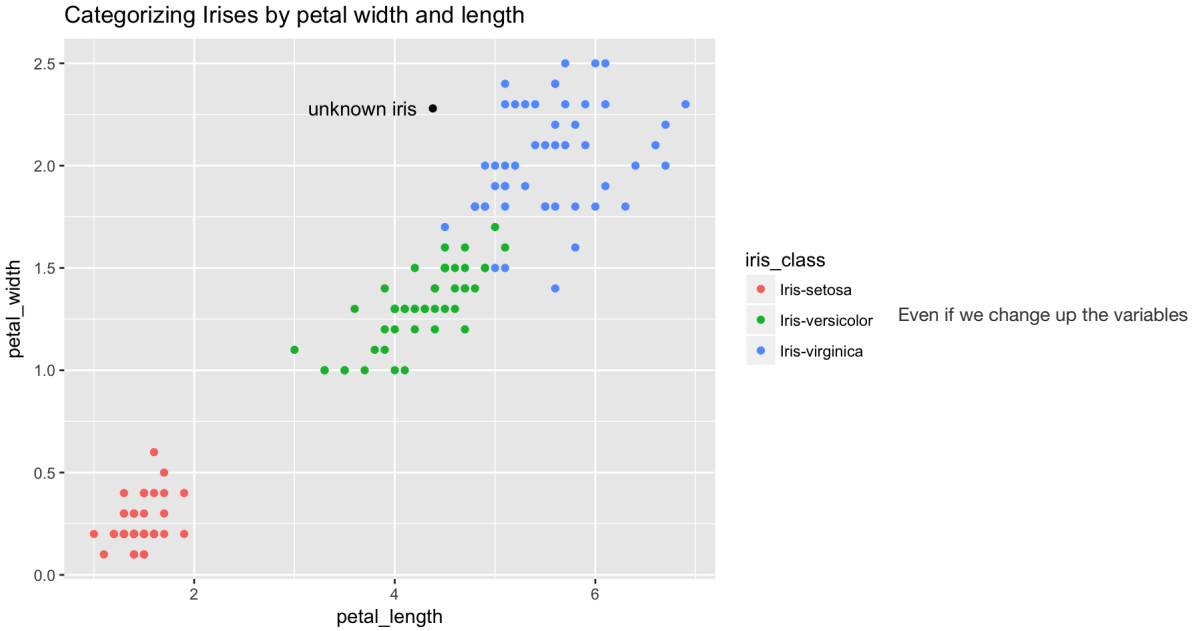
One idea is to graph the irises to see if we can tell them apart easily. Maybe we can split the graph into regions. Ideally, would be able to define clear boundaries for each iris so we can assign the unknown iris a category based on which region it falls under.

Let's try: Graphing

Let's try: Graphing

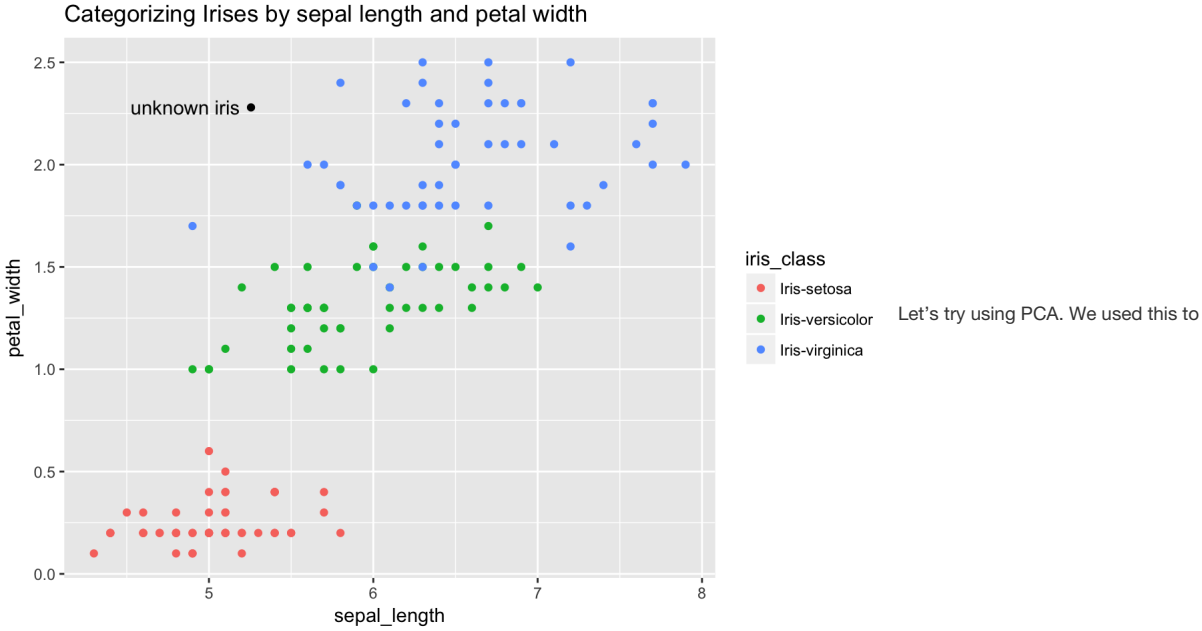
It's easy to tell if an iris is a setosa, but not if it is a versicolor or virginica.

```
#petal length & petal width
ggplot(iris_dat, aes(x = petal_length, y = petal_width)) +
  geom_point(aes(colour= iris_class)) +
  ggtitle("Categorizing Irises by petal width and length") +
  geom_point(data=u_iris, aes(x = petal_length, y = petal_width)) +
  geom_text(data=u_iris, aes(x = petal_length - 0.7, y = petal_width,
                             label="unknown iris"))
```



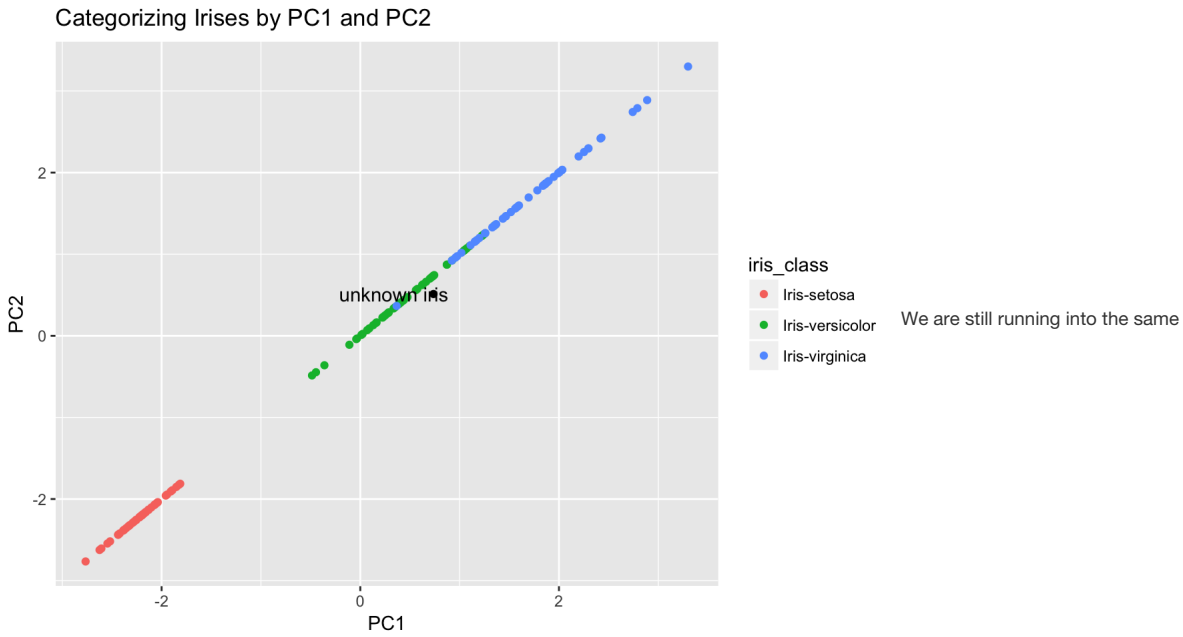
Even if we change up the variables displayed, it doesn't solve our problem of distinguishing between a versicolor or virginica easily by establishing clear regions with defined boundaries.

```
#sepal length & petal width
ggplot(iris_dat, aes(x = sepal_length, y = petal_width)) +
  geom_point(aes( colour= iris_class)) +
  ggtitle("Categorizing Irises by sepal length and petal width") +
  geom_point(data=u_iris, aes(x = sepal_length, y = petal_width)) +
  geom_text(data=u_iris, aes(x = sepal_length - 0.4, y = petal_width,
                             label="unknown iris"))
```



Let's try using PCA. We used this to solve our ranking problems, so maybe by calculating the PC scores of the irises to retain their variation, we can find separate the versicolor and the virginica points.

```
# making a new table without the iris class
iris_lim <- iris_dat %>% select(sepal_length, sepal_width, petal_length,
                                petal_width)
#performing the PCA
pca_values <- prcomp(x=iris_lim, scale. = TRUE)

#get the associated weights used to calculate PC1 and PC2
weights = pca_values$rotation[, 1:2]

#calculate the pc scores of the unknown
unknown_pc = predict(pca_values, u_iris)

#add the scores to u_iris
u_iris$PC1 = unknown_pc[1]
u_iris$PC2 = unknown_pc[2]

#making a new table with PC1, PC2 and their corresponding iris class
iris_pca <- data.frame("PC1" = pca_values$x[,1], "PC2" = pca_values$x[,1],
                       "iris_class" = iris_dat$iris_class)
#graph the result of PC1 vs PC2
ggplot(iris_pca, aes(x = PC1, y = PC2)) +
  geom_point(aes(colour= iris_class)) +
  ggtitle("Categorizing Irises by PC1 and PC2") +
  geom_point(data=u_iris, aes(x = PC1, y = PC2)) +
  geom_text(data=u_iris, aes(x = PC1 - 0.4, y = PC2,
                             label="unknown iris"))
```

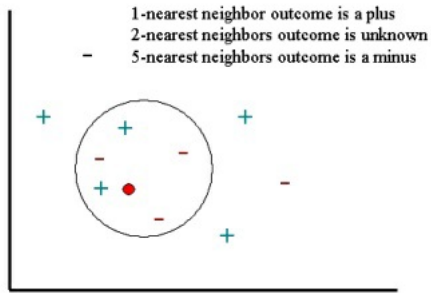


problem of overlap between the versicolor and the virginica. This is because PCA, at its core, is taking linear combinations of the 4 variables. And it turns out that versicolor and virginica are not linearly separable. So are we completely stuck?

Use K-nearest neighbors

What if we keep the idea of regions, but ignore the idea of boundaries? In other words, what if we look which type of irises that our unknown flower is close to? This is the basic idea of the k-nearest neighbors algorithm.

```
#download a visual example of k-nearest neighbors (optional)
k_nearest_pic_url <- "http://www.statsoft.com/portals/0/Support/KNNOverViewImageA.jpg"
download.file(k_nearest_pic_url, 'k_nearest_pic.jpg')
```



Example of k-nearest neighbors

In above example, we want to classify the red dot as either a plus or a minus. If we look at the closest point to the red dot, we can see that it is a plus. This is equivalent to using a `k = 1` neighbor to classify the red dot as a plus. If we use `k = 5` neighbors, then we look at the five closest

points (handily circled in the image). We can see that there are more minus points, so we would classify the red dot as a minus.

There are a couple of different methods for choosing a value for a k. The simplest one requires a separate validation set that lets us play with different values of k to see which ones gives us the lowest number of errors. To keep things simple in this blog post, we will just use a value of k = 4.

Trying K-nearest neighbors using 2 variables

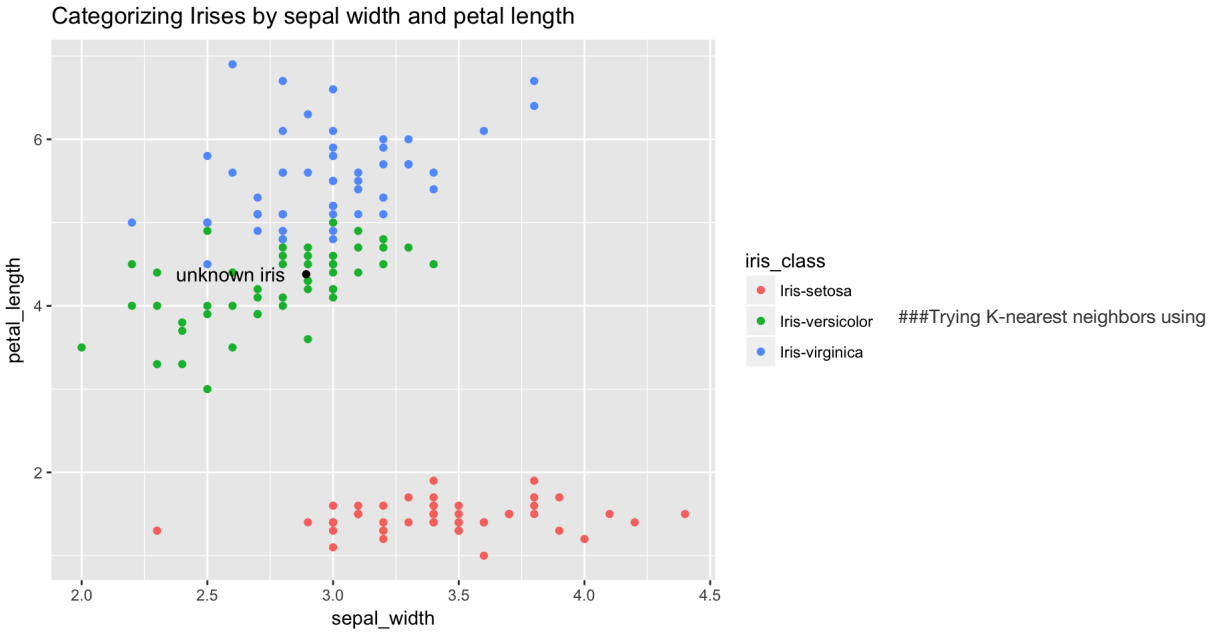
Let's use sepal width and petal length for a 2 dimensional view to get some intuition on how this works.

```
#define a function that calculates distances between 2 vectors
distance <- function(x, y) {
  return(sqrt(sum((x - y) ** 2)))
}

# select only the sepal width and petal length
small_unknown = c(unknown_iris[2], unknown_iris[3])

all_dist = c()
for (i in c(1:150)) { #for every flower that's in our dataset
  # find sepal width and petal length
  known_iris = c(iris_dat[[i, 2]], iris_dat[[i, 3]])
  #calcuate its distance to our unknown
  all_dist = c(all_dist, distance(small_unknown, known_iris))
}
#make a copy of our dataset then add all the distances as a new variable
iris_with_dist = iris_dat
iris_with_dist$all_dist = all_dist

#sort by the length of the distances and find the most common neighbor
sorted = iris_with_dist[order(iris_with_dist$all_dist),]
sort(table(sorted$iris_class[1:4]), decreasing=TRUE)[1:1]
```

```
## Iris-versicolor
##               4
```

We get that our unknown iris is a versicolor based on sepal length and petal width. This makes a lot of sense; although there aren't clear boundaries between versicolor and virginica, the unknown iris point is visually closer to the versicolor points more than the virginica points.

```
#sepal width & petal length
ggplot(iris_dat, aes(x = sepal_width, y = petal_length)) +
  geom_point(aes( colour= iris_class)) +
  ggtitle("Categorizing Irises by sepal width and petal length") +
  geom_point(data=u_iris, aes(x = sepal_width, y = petal_length)) +
  geom_text(data=u_iris, aes(x = sepal_width - 0.3, y = petal_length,
                             label="unknown iris"))
```



###Trying K-nearest neighbors using all 4 variables

```
all_dist = c()
for (i in c(1:150)) { #for every flower that's in our dataset
  # find their measurements
  known_iris = c()
  for (measure in c(1:4)) {
    known_iris = c(known_iris, iris_dat[[i, measure]])
  }

  #calcuate its distance to our unknown
  all_dist = c(all_dist, distance(unknown_iris, known_iris))
}
#make a copy of our dataset then add all the distances as a new variable
iris_with_dist = iris_dat
iris_with_dist$all_dist = all_dist

#sort by the length of the distances and find the most common neighbor
sorted = iris_with_dist[order(iris_with_dist$all_dist),]
sort(table(sorted$iris_class[1:4]), decreasing=TRUE)[1:1]
```

```
## Iris-versicolor
##               2
```

We get that the unknown iris should be veriscolor, just like before. However, there is a problem here. Since k-nearest neighbors uses euclidean distance, a variable or measurement that is not scaled properly or variables that are highly correlated could skew the distance calculations. This is a problem that we can solve using PCA.

K-nearest neighbors using PCA

We've already calculated the pc scores of our unknown iris and of the known irises. So we will use those scores to classify our unknown iris.

```
#pc scores of our unknown iris; it's called unknown_pc

all_dist = c()
for (i in c(1:150)) { #for every flower that's in our dataset
  # find their measurements
  known_iris = c()
  for (measure in c(1:2)) {
    known_iris = c(known_iris, iris_pca[[i, measure]])
  }

  #calcuate its distance to our unknown
  all_dist = c(all_dist, distance(unknown_pc, known_iris))
}
#make a copy of our dataset then add all the distances as a new variable
iris_with_dist = iris_dat
iris_with_dist$all_dist = all_dist

#sort by the length of the distances and find the most common neighbor
sorted = iris_with_dist[order(iris_with_dist$all_dist),]
sort(table(sorted$iris_class[1:4]), decreasing=TRUE)[1:1]
```

```
## Iris-versicolor
##               4
```

With this adjustment, we can conclude that for k=4 nearest neighbors our unknown iris is a versicolor.

Conclusion

Throughout this course, we've learned a lot of different techniques to clean and display data and then we generally rank them. This post looks at something else we can do with our data: solve a classification problem. We looked at the k-nearest neighbors algorithm and how it can be useful when we are working with datasets that are not linearly separable as well as how we can combine this algorithm with PCA to make up for its shortcomings.

Sources

- I got the data and the data dictionary from the UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/Iris
- Example of how it's used in machine learning. https://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#statug_cluster_sect027.htm
- Provided some inspiration, but doesn't go into the depth that I need. https://rstudio-pubs-static.s3.amazonaws.com/205883_b658730c12d14aa6996fe2f6c612c65f.html
- Source for the iris diagram picture. https://www.fs.fed.us/wildflowers/beauty/iris/images/blueflagiris_flower_lg.jpg
- A stack exchange post that explains how PCA works: https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues
  -Visual example of k-nearest neighbors source. http://www.statsoft.com/portals/0/Support/KNNOverViewImageA.jpg
- Source for a simple explanation of k-nearest neighbors. https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/
- A more in-depth look at k-nearest neighbors. https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/
- Dangers and common pitfalls of k nearest neighbors. https://mathbabe.org/2013/04/04/k-nearest-neighbors-dangerously-simple/
- K-nearest neighbors in relation to machine learning. https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/