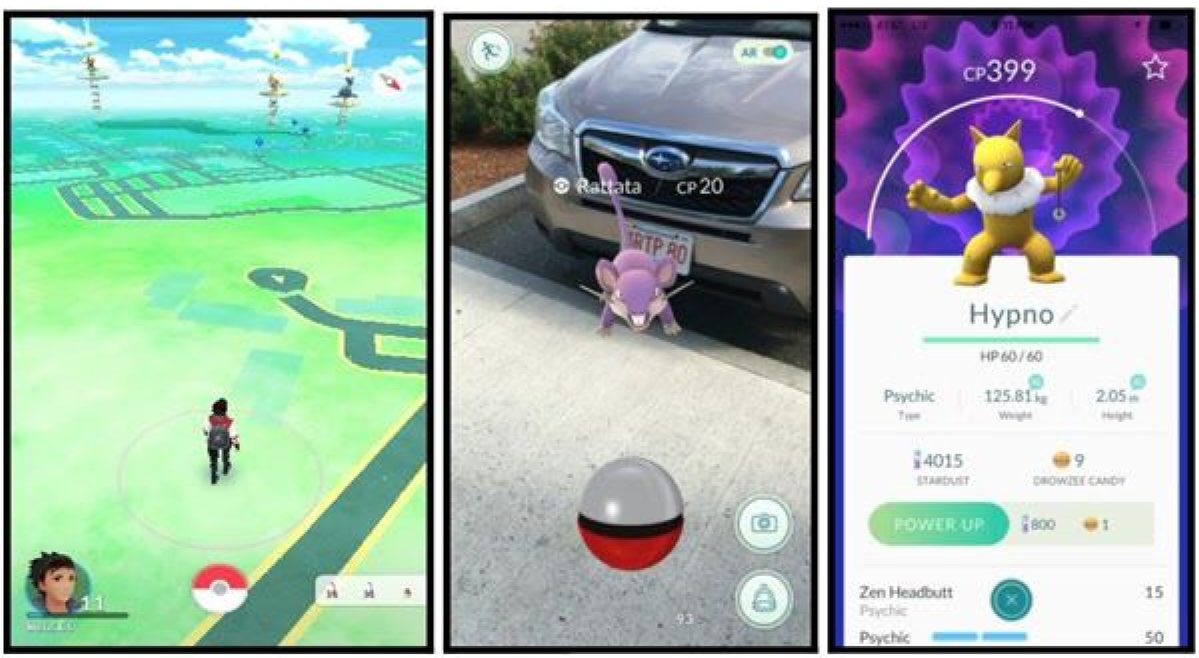
PokemonGo Data Analysis

Deanna Springgay
Lydia Savatsky
MA4790
December 2020

Abstract:

PokemonGo is a mobile augmented reality game developed by Niantic inc. for iOS, Android, and Apple Watch devices. It was initially released in selected countries in July of 2016. In the game, players use a mobile device's GPS capability to locate, capture, battle, and train virtual creatures, called Pokémon, who appear on the screen as if they were in the same real-world location as the player. A player can walk around anywhere in the world and catch pokemon. Below is a picture of what a player might see on their device when playing the game PokemonGo.



This paper will focus on predicting which pokemon will appear given various conditions such as weather, terrain type, time of day, type of area (urban, rual) and more. We have downloaded this data from Kaggle, a site that allows users to publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

Table of Contents

I. Background

Our dataset contains the sightings of Pokemon from Pokemon Go. The dataset was downloaded from kaggle, and the original goal of this dataset was to predict where a Pokemon will next appear. However, we have changed our goal to predict which Pokemon will appear given conditions other than location. We chose not to include location because we do not have to skillset to work with location data.

A. Description of Data

Before implementing pre-processing techniques, the data consisted of 296,021 observations and twenty eight predictors. Seven of these predictors were numeric while 22 were categorical. Below, we have included descriptions of the response variable and the predictors.

Response Variable:

pokemonId	Pokemon identifier

Categorical Predictors:

appearedTimeOfDay	time of the day of a sighting (night, evening, afternoon, morning)
appearedDayOfWeek	week day of a sighting (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)
terrainType	terrain where pokemon appeared described with help of GLCF Modis Land Cover (numeric value from 1-17)
closeToWater	did pokemon appear close (100m or less) to water (Boolean, same source as above)
weather	weather type during a sighting (Foggy Clear, PartlyCloudy, MostlyCloudy, Overcast, Rain, BreezyandOvercast, LightRain, Drizzle, BreezyandPartlyCloudy, HeavyRain, BreezyandMostlyCloudy, Breezy, Windy, WindyandFoggy, Humid, Dry, WindyandPartlyCloudy, DryandMostlyCloudy, DryandPartlyCloudy, DrizzleandBreezy, LightRainandBreezy, HumidandPartlyCloudy, HumidandOvercast, RainandWindy)
weatherIcon	a compact representation of the weather at the location of a sighting (fog, clear-night, partly-cloudy-night, partly-cloudy-day, cloudy, clear-day, rain, wind)

urban	how urban is location where pokemon appeared (Boolean, built on Population density)
suburban	how urban is location where pokemon appeared (Boolean, built on Population density)
midurban	how urban is location where pokemon appeared (Boolean, built on Population density)
rural	how rural is location where pokemon appeared (Boolean, built on Population density)
gymIn100m	is there a gym in 100 meters? (Boolean)
gymIn250m	is there a gym in 250 meters? (Boolean)
gymIn500m	is there a gym in 1000 meters? (Boolean)
gymIn1000m	is there a gym in 1000 meters? (Boolean)
gymIn2500m	is there a gym in 2500 meters? (Boolean)
gymIn5000m	is there a gym in 5000 meters? (Boolean)
pokestopIn100m	is there a pokestop in 100 meters? (Boolean)
pokestopIn250m	is there a pokestop in 250 meters? (Boolean)
pokestopIn500m	is there a pokestop in 1000 meters? (Boolean)
pokestopIn1000m	is there a pokestop in 1000 meters? (Boolean)
pokestopIn2500m	is there a pokestop in 2500 meters? (Boolean)
pokestopIn5000m	is there a pokestop in 5000 meters? (Boolean)

Numerical Predictors:

gymDistanceKm	how far is the nearest gym in km from a sighting. Boolean value for 100m,
pokestopDistanceKm	how far is the nearest pokestop in km from a sighting
Population density	what is the population density per square km of a sighting
temperature	temperature in celsius at the location of a sighting

windSpeed	speed of the wind in km/h at the location of a sighting
windBearing	wind direction
pressure	atmospheric pressure in bar at the location of a sighting

II. Pre-Processing the Data

A. Analyze Missing Values

Our first step in pre-processing the data was to analyze any missing values. We found that our dataset consisted of 39 missing values. Since this was such a small portion of the data, we chose to remove the missing values. This left us with 295,982 observations.

B. Creating Dummy Variables

We then created dummy variables for our 22 categorical variables. This left us with 100 total predictors. Out of these predictors, 92 were categorical dummy variables and 7 were numeric variables.

C. Removing Near Zero Variance

We then analyzed the predictors with near zero variance. We had a total of 43 out of 100 predictors with near zero variance. We removed these predictors from the data set, reducing the number of predictors to 57.

D. Center and Scale and Principle Component Analysis

Next, we centered and scaled our data to reduce the skewness of our variables. Because some of our predictors were highly correlated, we decided to implement the principle component analysis technique in order to reduce high correlation among predictors. A correlation plot of our 57 predictors is shown in Figure 1. This created thirty one principle components. Figure 2 below displays our numeric variables before implementing center, scale, and principle component analysis transformation. We can see that wind speed, population density, gym distance in Km, and pokestop distance in Km are highly skewed to the right. Figures 3 through 5 display the histograms after implementing center, scale, and principle component analysis transformations. We can see that a majority of the principle components are centered, with a few components containing some outliers still. We hope that a spatial sign transformation will reduce these outliers.

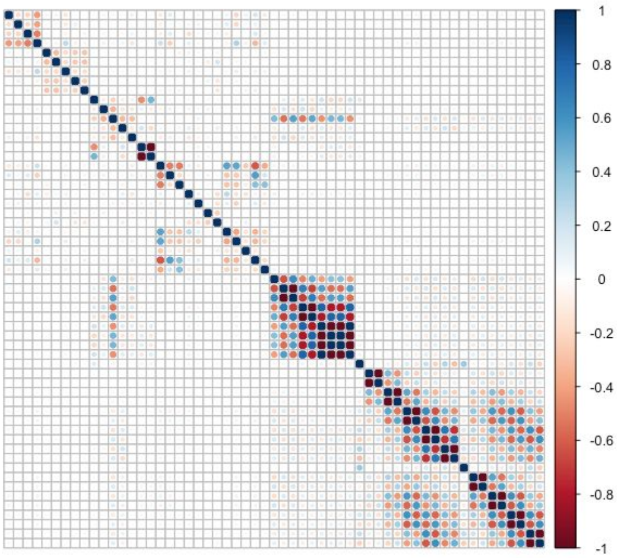


Figure 1: Correlation Plot Among 57 Predictors

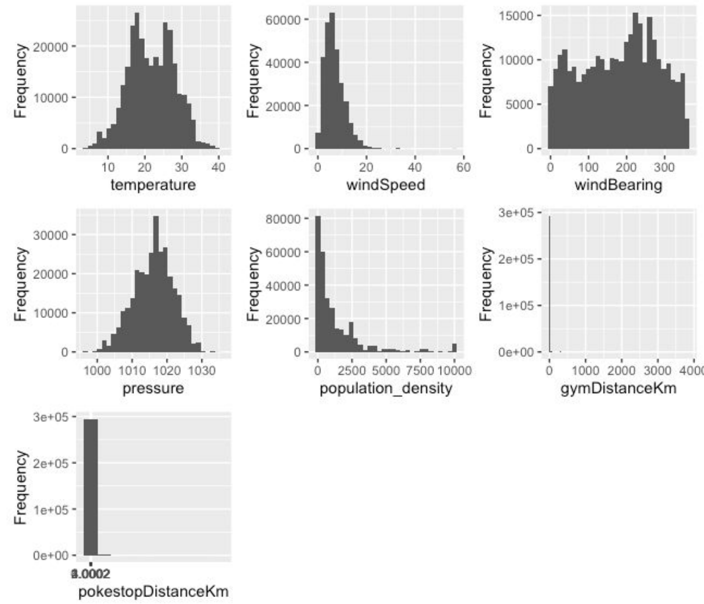


Figure 2: Numeric Variables <u>Before</u> Implementing Center, Scale, and PCA

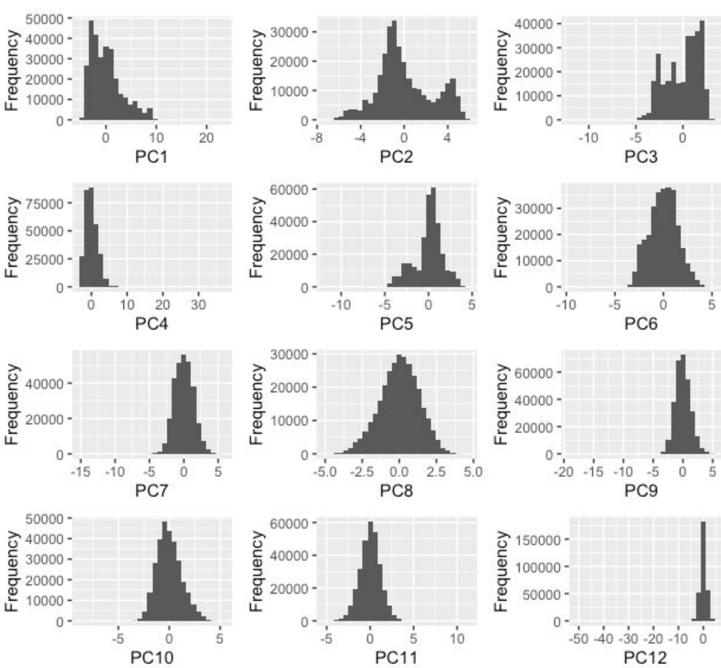


Figure 3: Components 1-12 <u>After</u> Implementing Center, Scale, and PCA

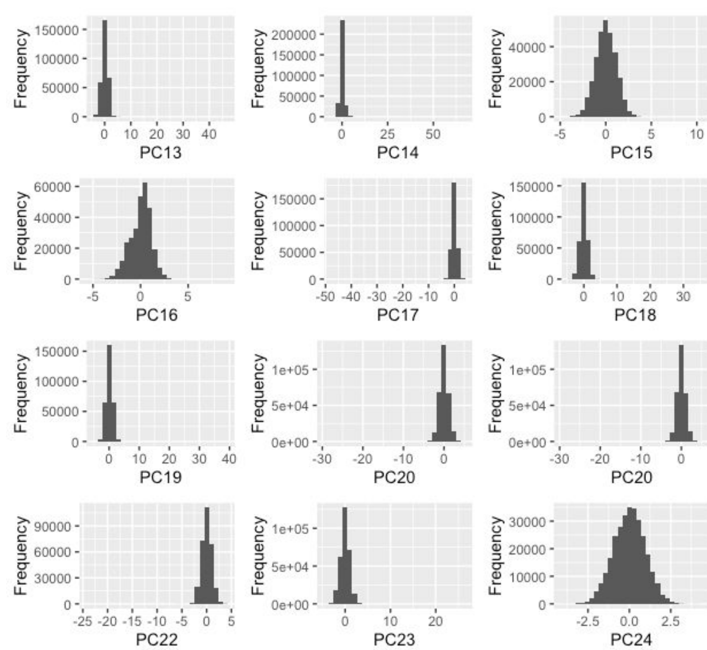


Figure 4: Components 13-24 <u>After</u> Implementing Center, Scale, and PCA

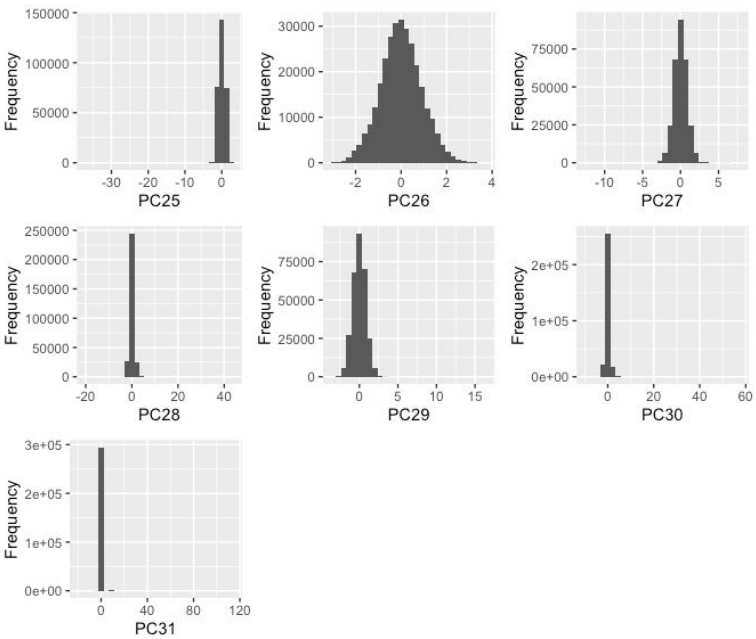


Figure 5: Components 25-31 <u>After</u> Implementing Center, Scale, and PC

E. Spatial Sign Transformation

In order to reduce outliers, we performed a spatial sign transformation. Figures 6 through 11 below display boxplots of each principle component before and after implementing a spatial sign transformation. We can see that a majority of the components no longer have outliers. For the few components that do still have outliers, the scale for the boxplots reduced, meaning that the number of outliers for these components reduced.

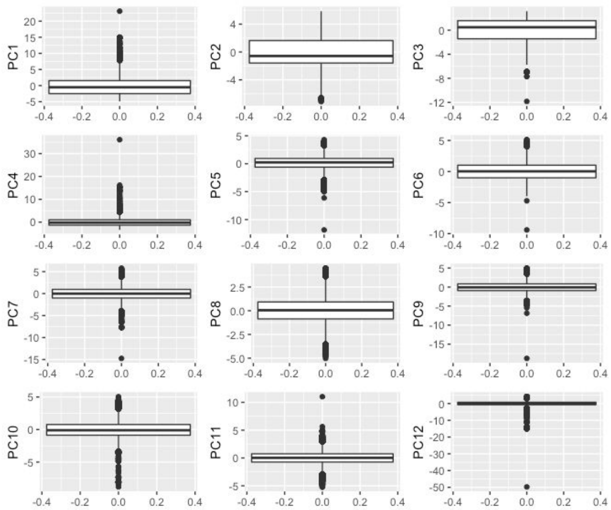


Figure 6: Components 1-12 <u>Before</u> Implementing Spatial Sign Transformation

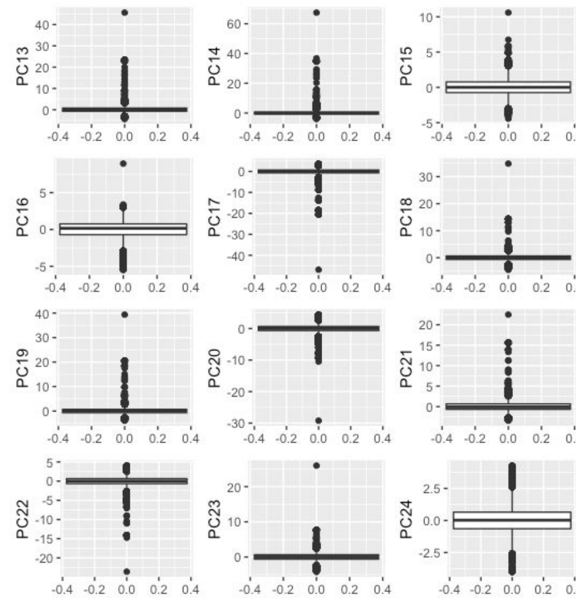


Figure 7: Components 13-24 <u>Before</u> Implementing Spatial Sign Transformation

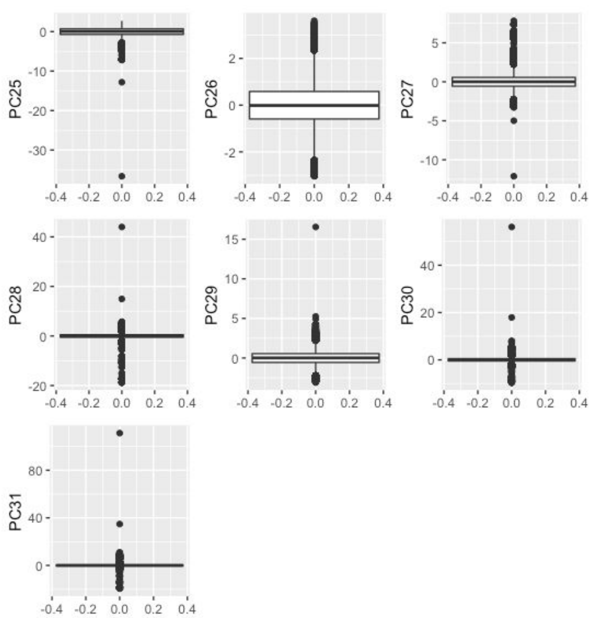


Figure 8: Components 25-31 <u>Before</u> Implementing Spatial Sign Transformation

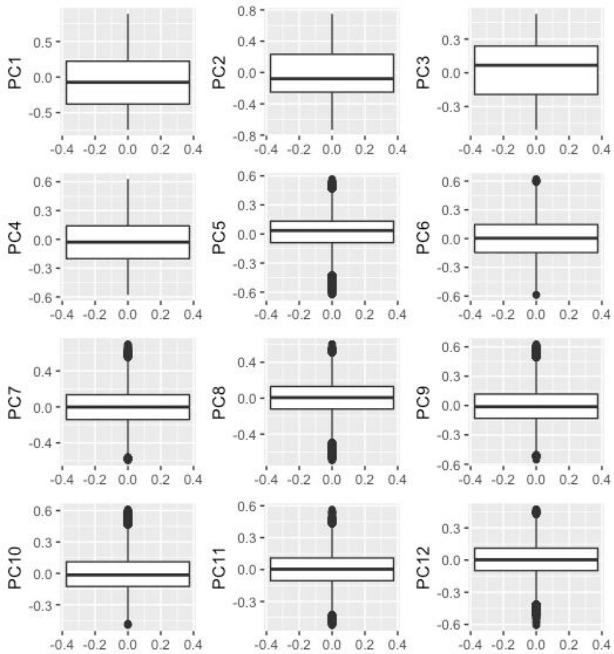


Figure 9: Components 1-12 <u>After</u> Implementing Spatial Sign Transformation

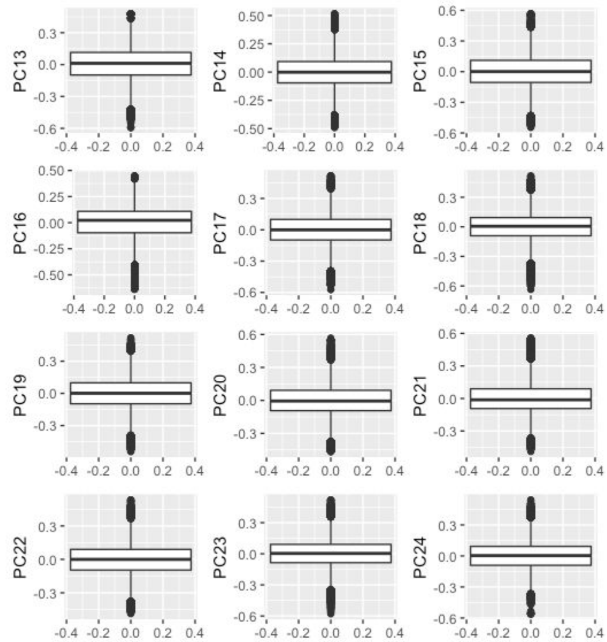


Figure 10: Components 13-24 <u>After</u> Implementing Spatial Sign Transformation

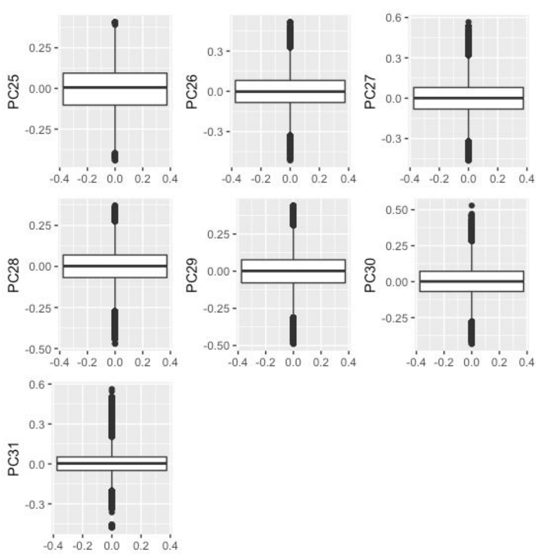


Figure 11: Components 1-12 <u>After</u> Implementing Spatial Sign Transformation

III. Adjusting the Response Variable Naming Conventions

Before we could begin training the data, we had to adjust the naming convention of our response variable. Our response variable was categorical, however, the variable was using number IDs to classify each pokemon. Since R does not allow for a string of numbers to be a factor, we renamed each ID to instead be the name of the pokemon. There were a total of 151 different pokemon IDs that were renamed.

IV. Splitting the Data

Figure 12 below displays the distribution of our response variable pokemonID. As we can see, the distribution is unbalanced, with some pokemonID's occurring as high as 50,000 times and some occurring as low as only one time. We will therefore split our dataset using a stratified random sampling technique.

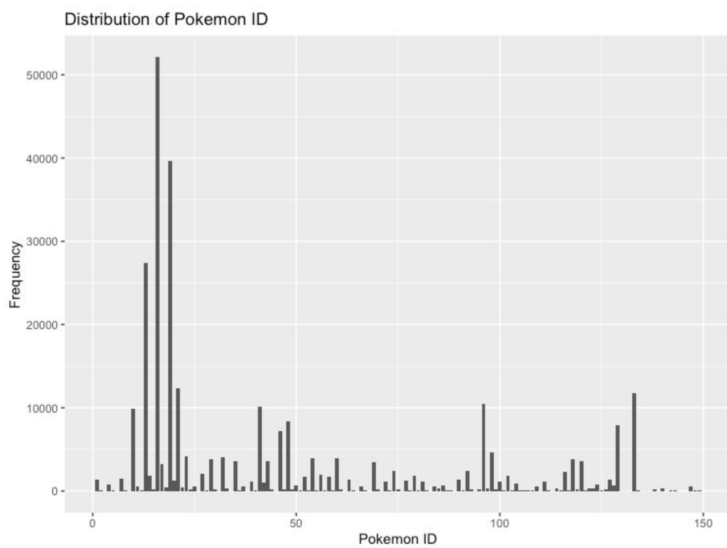


Figure 12: Distribution of PokemonID

Because our dataset is so large, we have decided to only run the models on a quarter of the data. This means that we will generate models on 73,996 observations rather than generating models on 295,982 observations. After subsetting the data into a fourth of the observations, we split up this data into a training and testing set using a stratified random sampling technique. We took 80% of the 73,996 observations for the training set and 20% of those observations for the testing set. We then ended up with 59,198 observations on the training set and 14,798 on the testing set.

V. Resampling Technique

We decided to use a 10-fold cross validation technique for our dataset. We believed that using cross validation was best for our large dataset. We hoped that this technique would reduce bias and increase precision within our models.

VI. Model Fitting

A. Linear Classification Models

Below is a table summarizing our findings after generating the four linear classification models on our training data. Since our response variable, pokemonID, has more than two classes, we will use the kappa statistic when determining which model is best. The kappa value does take into account class distributions rather than calculating the overall accuracy. The best linear model appears to be the Logistic Regression model. Although this model has the lowest accuracy rate, it does compute the highest kappa value between these four models.

These four models generated very low accuracy and kappa values. Perhaps some reasons behind these low values could be that our data is not linear or that our predictors are not sufficient in predicting the pokemon found.

Models	Tuning Parameter	Accuracy	Kappa
Logistic Regression	decay = 0.0001	0.1328164	0.02868753
Linear Discriminant Analysis	N/A	0.1664909	0.01440751
Partial Least Squares	ncomp = 20	0.1744347	.004443577
Penalized	alpha = 0 and lambda = 0.01	0.1750971	0.0004753644

B. Nonlinear Classification Models

Below is a table summarizing our findings after generating seven different nonlinear models. Since our data is extremely large, we ran Mixture Discriminant Analysis, Neural Networks, and Support Vector Machine models on only one percent of our data. This resulted in running the models on 2,368 observations for our training set. The testing set consisted of 592 observations. The best nonlinear model is the K-Nearest Neighbors model with a kappa value of 0.035. The Naive Bayes model is the second best nonlinear classification model with a kappa value of 0.028. These two models have two of the highest accuracy rates between the seven nonlinear models. We have decided that these two nonlinear models perform the best for our data. Therefore, we will proceed by predicting the Naive Bayes model and the KNN model on our testing set.

These seven models generated very low accuracy and kappa values, just as the linear models had done. Therefore, the reason for these low values is not that our data is not linear. Instead, perhaps the reason behind these low values is that our predictors are not sufficient in predicting the pokemon found. Another possible reason could be that the pokemon are found randomly, and one cannot predict which pokemon will be found when given conditions such as terrain type, weather, temperature, time of day, and more.

Models	**Tuning Parameter**	**Accuracy**	**Kappa**
Regularized Discriminant Analysis	gamma = 1 and lambda = 5	0.0343	0
Mixture Discriminant Analysis (1% of data)	subclasses = 5	0.04572785	0.0025162158
Neural Networks (1% of data)	size = 12 and decay = 0.1	0.005994365	0.002220065
Flexible Discriminant Analysis	Degree = 1 Nprune = 2	0.1762636	0
Support Vector Machines (1% of data)	sigma = 0.01269174 and C = 1	0.004935692	0.0004591553
K-Nearest Neighbors	k = 15	0.15059988	0.03494705
Naive Bayes	fL = 2 Usekernel = TRUE Adjust = TRUE	0.1420867	0.02820561

C. Model Selecting

Below is a table summarizing our findings after predicting the KNN and Naive Bayes models on our testing data. The KNN model calculated a negative value, meaning that the model is predicting in the opposite direction of truth. We have determined that the best performing model was the Naive Bayes model, as it generated the highest kappa value between the two models.

Models	Tuning Parameter	Accuracy	Kappa
K-Nearest Neighbors	k = 15	0.005930690	-0.002002549
Naive Bayes	fL = 2 Usekernel = TRUE Adjust = TRUE	0.14862448	0.03240625

Figure 13 below displays the confusion matrix predicted on the testing side for the Naive Bayes model.

Confusion Matrix and Statistics

Prediction \ Reference	Abra	Aerodactyl	Alakazam	Arbok	Arcanine	Beedrill	Bellsprout	Blastoise	Bulbasaur	Butterfree	Caterpie	Chansey	Charizard	Charmander	Charmeleon	Clefable	Clefairy	Cloyster	Cubone	Dewgong	Diglett	Dodio	Doduo
Abra	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Aerodactyl	2	0	0	0	0	0	9	0	1	0	29	0	0	2	0	0	13	0	4	0	1	0	1
Alakazam	0	0	0	0	0	0	0	0	0	0	6	5	0	0	0	0	0	0	0	0	0	0	2
Arbok	8	0	0	2	0	0	3	0	0	0	4	0	0	0	0	0	4	0	5	0	2	0	0
Arcanine	3	0	0	0	0	0	0	0	0	1	6	0	0	0	0	0	0	0	0	0	0	0	0
Beedrill	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Prediction \ Reference	Dragonair	Dragonite	Dratini	Drowzee	Dugtrio	Eevee	Ekans	Electabuzz	Electrode	Exeggcute	Exeggutor	Fearow	Flareon	Gastly	Gengar	Geodude	Gloom	Golbat	Goldeen	Golduck	Golem	Graveler	Grimer	Growlithe	Gyarados
Abra	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	4	0	0	0	0	0	1	0	0	0
Aerodactyl	1	0	0	23	0	14	8	1	0	6	0	2	0	3	0	9	0	4	8	0	0	0	0	1	0
Alakazam	0	0	0	0	0	2	3	0	0	0	0	0	0	0	0	11	0	0	1	0	0	0	0	1	0
Arbok	0	0	0	4	0	9	22	0	0	0	0	0	0	0	0	11	0	1	1	1	0	0	0	10	0
Arcanine	0	0	0	1	0	2	3	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Beedrill	0	0	0	3	0	5	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Prediction \ Reference	Haunter	Hitmonchan	Hitmonlee	Horsea	Hypno	Ivysaur	Jigglypuff	Jolteon	Jynx	Kabuto	Kabutops	Kadabra	Kakuna	Kangaskhan	Kingler	Koffing	Krabby	Lapras	Lickitung	Machoke	Machop	Magikarp	Magmar	Magnemite	Magnetron
Abra	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Aerodactyl	3	2	0	2	0	0	4	0	4	0	0	0	0	0	1	0	4	0	0	0	2	24	3	0	0
Alakazam	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
Arbok	0	0	0	3	0	0	0	0	0	0	0	0	2	0	1	1	0	0	0	0	0	3	0	0	0
Arcanine	0	0	0	4	0	0	0	0	0	0	0	0	0	0	1	0	7	0	0	0	0	0	0	0	0
Beedrill	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Prediction \ Reference	Mankey	Marowak	Meowth	Metapod	MrMime	Muk	Nidoking	Nidoqueen	NidoranF	NidoranM	Nidorina	Nidorino	Ninetales	Oddish	Omanyte	Omastar	Onix	Paras	Parasect	Persian	Pidgeot	Pidgeotto	Pidgey	Pikachu	Pinsir	Poliwag
Abra	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	1	15	0	0	0
Aerodactyl	7	0	8	1	0	0	0	0	5	9	0	0	0	3	0	0	0	13	1	0	1	0	78	0	1	21
Alakazam	4	0	3	0	0	0	0	0	4	0	0	0	0	1	0	0	0	3	0	0	0	2	40	0	0	2
Arbok	7	0	2	1	0	0	0	0	4	1	0	0	0	0	1	0	0	12	0	0	0	2	48	0	0	1
Arcanine	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	11	0	0	0	0	25	0	0	0
Beedrill	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	4	0	0	0	0	12	0	0	0

Prediction \ Reference	Poliwhirl	Poliwrath	Ponyta	Porygon	Primeape	Psyduck	Raichu	Rapidash	Raticate	Rattata	Rhydon	Rhyhorn	Sandshrew	Sandslash	Scyther	Seadra	Seaking	Seel	Shellder	Slowbro	Slowpoke	Snorlax	Spearow	Squirtle	Starmie
Abra	0	0	1	0	0	0	0	0	0	8	0	1	3	0	0	0	0	0	1	0	0	0	0	0	0
Aerodactyl	0	0	1	0	0	6	0	0	1	47	0	2	5	0	2	0	0	0	9	0	5	1	27	4	2
Alakazam	0	0	3	0	0	0	0	0	2	15	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
Arbok	0	0	4	0	0	0	0	0	0	10	0	2	14	0	0	0	0	0	0	0	0	0	8	1	0
Arcanine	0	0	0	0	0	0	0	0	0	12	0	8	1	0	0	0	0	0	0	0	0	0	8	0	0
Beedrill	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0	1	0	0	0	2	0	0

Prediction \ Reference	Staryu	Tangela	Tantacruel	Tauros	Tentacool	Vaporeon	Venomoth	Venonat	Venusaur	Victreebel	Vileplume	Voltorb	Vulpix	Wartortle	Weedle	Weepinbell	Weezing	Wigglytuff	Zubat
Abra	0	0	0	0	0	0	0	5	0	0	0	0	0	0	2	0	0	0	0
Aerodactyl	12	0	2	0	1	0	0	7	0	0	0	0	0	0	45	0	0	0	30
Alakazam	0	2	0	0	0	0	0	2	0	0	0	0	0	0	6	0	0	0	8
Arbok	0	0	0	2	0	0	0	3	0	0	0	3	0	0	24	0	0	0	5
Arcanine	0	5	0	0	0	0	0	2	0	0	0	0	1	0	8	0	0	0	1
Beedrill	1	0	0	0	0	0	0	2	0	0	0	0	0	0	4	0	0	0	0

[reached getOption("max.print") -- omitted 137 rows]

Figure 13: Confusion Matrix on Naive Bayes Model

Figure 14 below shows the important predictors for the Naive Bayes model. Unfortunately, our second attempt to re-build the model took over three days and did not complete within this assignment's window. Ideally, we would have a simpler representation of the important predictors such that they would be legible and easier to interpret their overall importance.

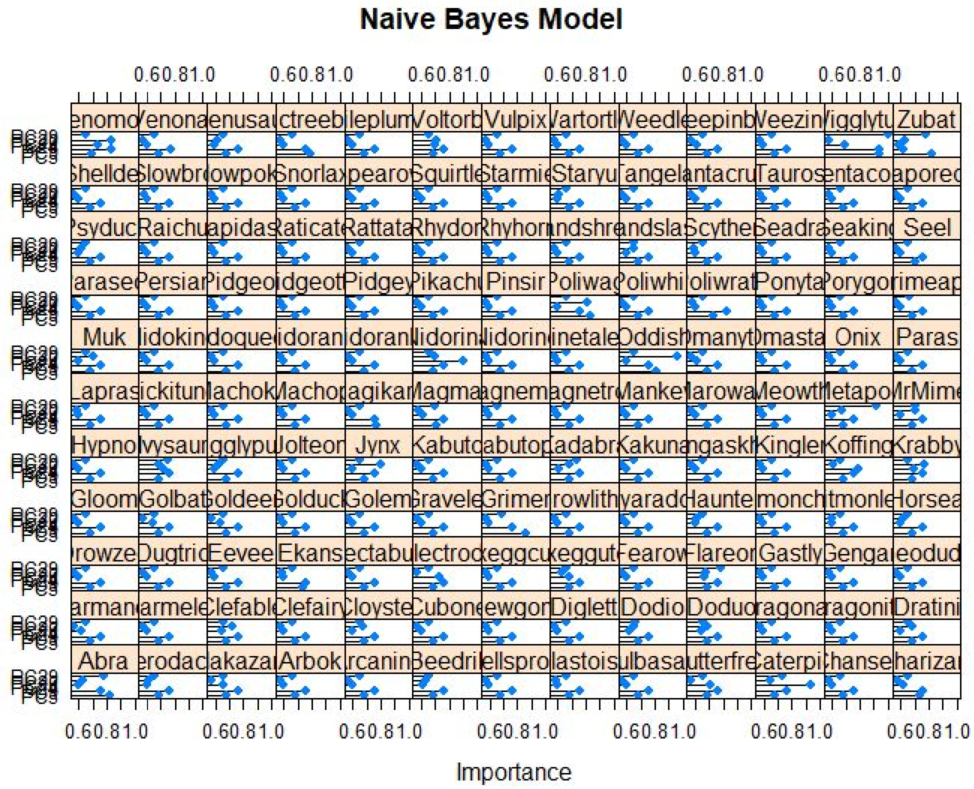


Figure 14: Important Predictors of the Naive Bayes Model

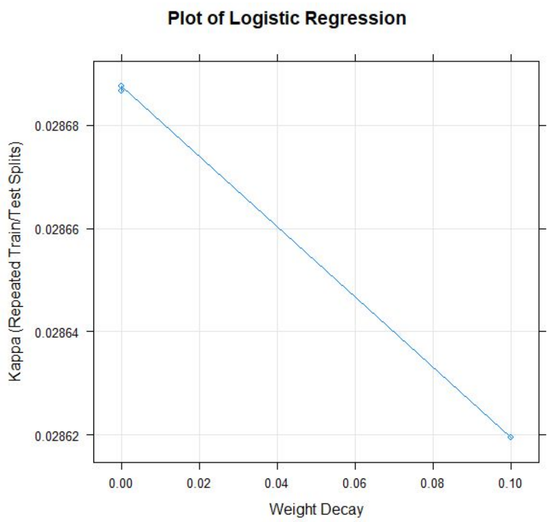
VII. Summary

In order to predict which pokemon would appear given conditions such as weather, temperature, pressure, terrain type, and more, we gathered the data, pre-processed our data, renamed our response variable, created a training and testing set for our data, and performed linear and nonlinear classification models on the data. Before pre-processing, we had 296,021 observations and 30 predictors (7 numerical and 22 categorical). While preprocessing the data, we (1) removed missing values, (2) created dummy variables, (3) removed near zero variance, (4) centered, scaled, and implemented principle component analysis transformations, and finally (5) implemented a spatial sign transformation. Pre-processing the data left us with 295,982 observations and 31 principle component predictors. After pre-processing, we renamed our response variables so that they were no longer a string of numbers but rather the name of the pokemon. We then separated our data into a training and testing set. Since our data was extremely large, we decided to perform our models on a quarter of the data, resulting in 59,198 observations on the training set and 14,798 on the testing set. Finally, we performed four linear models and seven nonlinear classification models. The best performing model was the Naive Bayes model, so we predicted this model on the testing set of our data and found a kappa value of 0.0324.

Appendix 1: Supplemental Material for Linear Classification Models

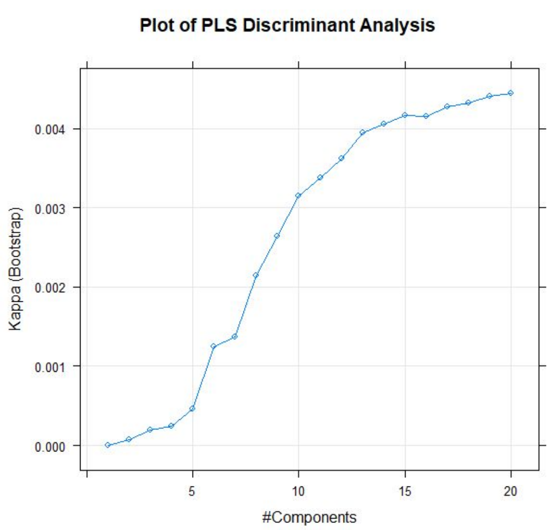
Logistic Regression Model

The optimum logistic regression model selected a weight decay of 0.0001.



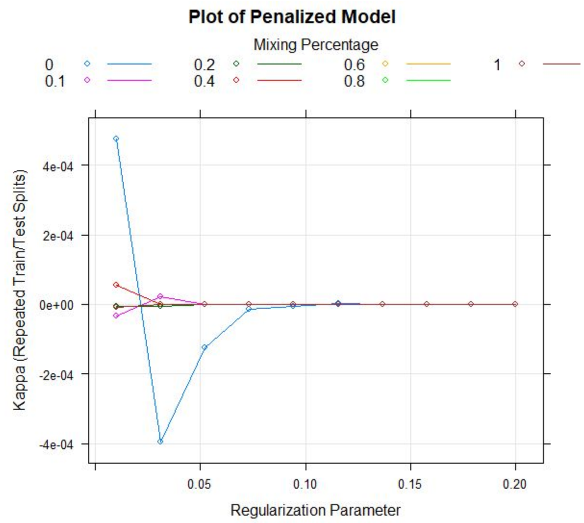
Partial Least Squares Model

The optimum partial least squares model selected 20 components.



Penalized Model

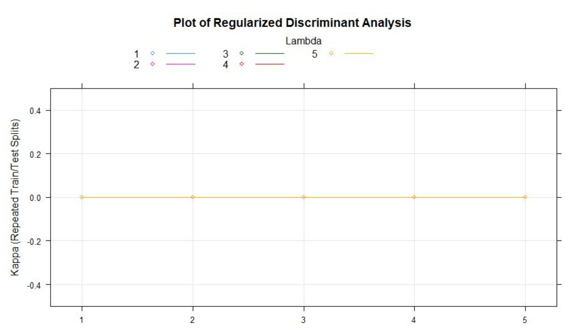
The tuning parameters for the penalized model are alpha and lamda. The optimum penalized model selected an alpha of zero and a lambda of 0.01.



Appendix 2: Supplemental Material for Nonlinear Classification Models

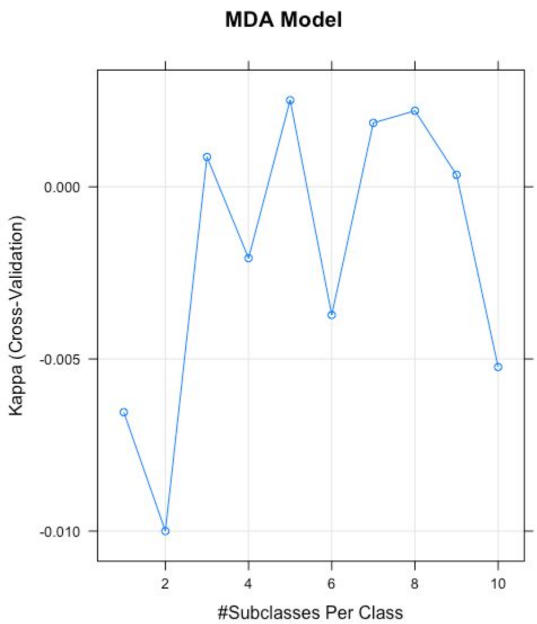
Regularized Discriminant Analysis

The tuning parameters for the regularized discriminant analysis model are lambda and gamma. The optimum regularized discriminant analysis model selected a lamda of 5 and a gamma of 1.



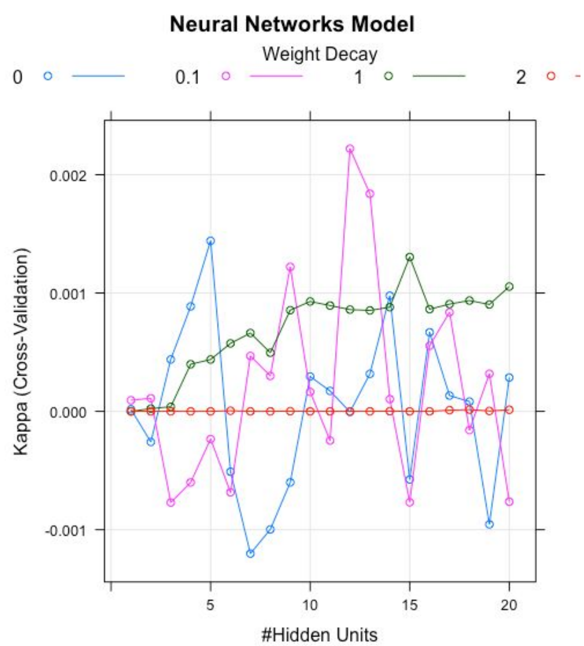
Mixture Discriminant Analysis

The optimum mixture discriminant analysis model selected 5 subclasses.



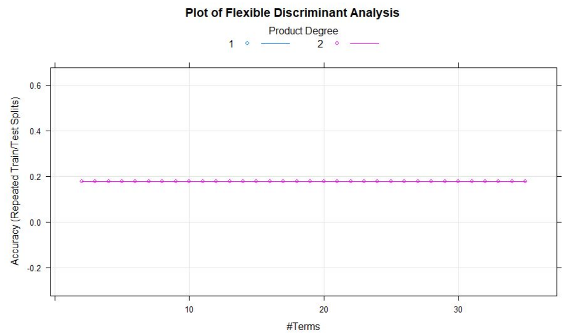
Neural Networks

The tuning parameters for the neural networks model are size and decay. The optimum neural networks model selected a size of 12 and a decay of 0.1.



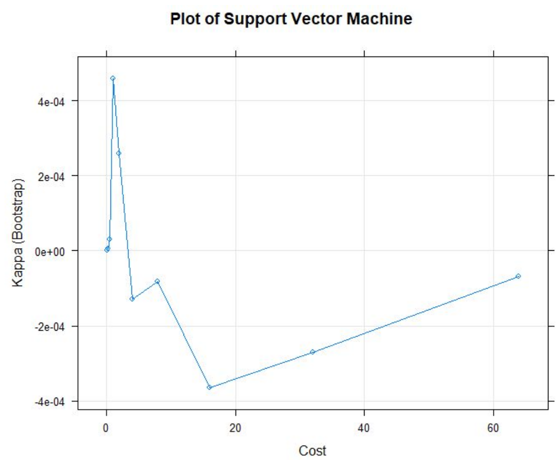
Flexible Discriminant Analysis

The tuning parameters for the flexible discriminant analysis model are degree and number of terms. The optimum flexible discriminant analysis model selected a degree of 1 and 2 terms.



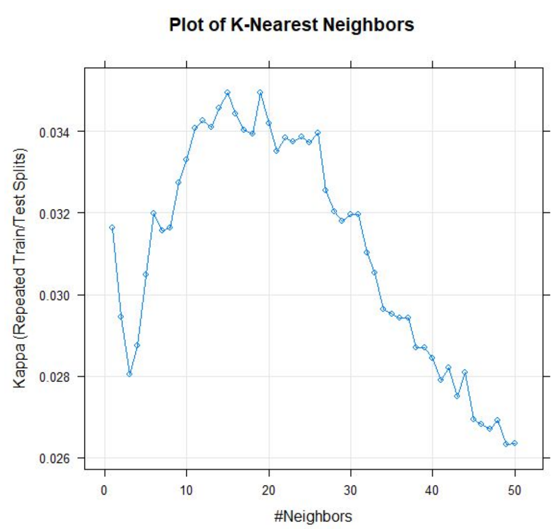
Support Vector Machines

The tuning parameters for support vector machines are cost and sigma. The optimum support vector machines model selected a cost of 1 and a sigma of 0.0127.



K-Nearest Neighbors

The optimum K-Nearest Neighbors model selected 15 neighbors.



R Code:

```
library(AppliedPredictiveModeling)
library(caret)
library(earth)
library(e1071)
library(kernlab)
library(tidyverse)

# PreProcessing the Pokemon Data --------------------------------------------
# Pokemon is the name of the original dataset
# start with 208 columns

# Selecting Columns and Remove NAs
kept_pokemon <- Pokemon[, c(1, 13, 16, 20, 21, 24:29, 38:56)]
kept_pokemon <- kept_pokemon %>%
  drop_na() %>%
  mutate_at(vars(terrainType), as.character)
colnames(kept_pokemon)
response_pokemon <- kept_pokemon[, 1]
response_pokemon <- as.data.frame(response_pokemon)
dirty_pokemon <- kept_pokemon[, -1]
# columns: 30

# Dummy Variables
dmy <- dummyVars("~.", data = dirty_pokemon)
dummy_pokemon <- data.frame(predict(dmy, newdata = dirty_pokemon))
# columns: 100
```

```
# Near Zero Variance
zed <- nearZeroVar(dummy_pokemon)
non_zed_pokemon <- dummy_pokemon[, -zed]
# columns: 57

# Correlation plot
correlations <- cor(non_zed_pokemon)
corrplot(correlations, tl.pos = "n")

# Histograms before center,scale,PCA
histogram_temperature <- non_zed_pokemon %>%
  ggplot(aes(temperature)) +
  geom_histogram() +
  ylab("Frequency")

histogram_windSpeed <- non_zed_pokemon %>%
  ggplot(aes(windSpeed)) +
  geom_histogram() +
  ylab("Frequency")

histogram_windBearing <- non_zed_pokemon %>%
  ggplot(aes(windBearing)) +
  geom_histogram() +
  ylab("Frequency")

histogram_pressure <- non_zed_pokemon %>%
  ggplot(aes(pressure)) +
  geom_histogram() +
  ylab("Frequency")

histogram_population_density <- non_zed_pokemon %>%
  ggplot(aes(population_density)) +
  geom_histogram() +
  ylab("Frequency")

histogram_gymDistanceKm <- non_zed_pokemon %>%
  ggplot(aes(gymDistanceKm)) +
  geom_histogram(binwidth = 50) +
  ylab("Frequency")
```

```
histogram_pokestopDistanceKm <- non_zed_pokemon %>%
  ggplot(aes(pokestopDistanceKm)) +
  geom_histogram(binwidth = 50) +
  scale_x_continuous(breaks = seq(0.0002, 5, 1)) +
  ylab("Frequency")

grid.arrange(histogram_temperature, histogram_windSpeed, histogram_windBearing,
             histogram_pressure, histogram_population_density, histogram_gymDistanceKm,
             histogram_pokestopDistanceKm)


# Center, Scale, PCA
scaled_centered <- preProcess(non_zed_pokemon, method = c("center", "scale", "pca"))
scaled_centered_pokemon <- predict(scaled_centered, non_zed_pokemon)
# columns: 31

# Histograms after center,scale,PCA
histogram_PC1 <- scaled_centered_pokemon %>%
  ggplot(aes(PC1)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC2 <- scaled_centered_pokemon %>%
  ggplot(aes(PC2)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC3 <- scaled_centered_pokemon %>%
  ggplot(aes(PC3)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC4 <- scaled_centered_pokemon %>%
  ggplot(aes(PC4)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC5 <- scaled_centered_pokemon %>%
  ggplot(aes(PC5)) +
  geom_histogram() +
```

```
  ylab("Frequency")

histogram_PC6 <- scaled_centered_pokemon %>%
  ggplot(aes(PC6)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC7 <- scaled_centered_pokemon %>%
  ggplot(aes(PC7)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC8 <- scaled_centered_pokemon %>%
  ggplot(aes(PC8)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC9 <- scaled_centered_pokemon %>%
  ggplot(aes(PC9)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC10 <- scaled_centered_pokemon %>%
  ggplot(aes(PC10)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC11 <- scaled_centered_pokemon %>%
  ggplot(aes(PC11)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC12 <- scaled_centered_pokemon %>%
  ggplot(aes(PC12)) +
  geom_histogram() +
  ylab("Frequency")

grid.arrange(histogram_PC1, histogram_PC2, histogram_PC3, histogram_PC4, histogram_PC5,
histogram_PC6,
```

```
            histogram_PC7, histogram_PC8, histogram_PC9, histogram_PC10, histogram_PC11,
histogram_PC12)

histogram_PC13 <- scaled_centered_pokemon %>%
  ggplot(aes(PC13)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC14 <- scaled_centered_pokemon %>%
  ggplot(aes(PC14)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC15 <- scaled_centered_pokemon %>%
  ggplot(aes(PC15)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC16 <- scaled_centered_pokemon %>%
  ggplot(aes(PC16)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC17 <- scaled_centered_pokemon %>%
  ggplot(aes(PC17)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC18 <- scaled_centered_pokemon %>%
  ggplot(aes(PC18)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC19 <- scaled_centered_pokemon %>%
  ggplot(aes(PC19)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC20 <- scaled_centered_pokemon %>%
  ggplot(aes(PC20)) +
```

```
  geom_histogram() +
  ylab("Frequency")

histogram_PC21 <- scaled_centered_pokemon %>%
  ggplot(aes(PC20)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC22 <- scaled_centered_pokemon %>%
  ggplot(aes(PC22)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC23 <- scaled_centered_pokemon %>%
  ggplot(aes(PC23)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC24 <- scaled_centered_pokemon %>%
  ggplot(aes(PC24)) +
  geom_histogram() +
  ylab("Frequency")

grid.arrange(histogram_PC13, histogram_PC14, histogram_PC15, histogram_PC16,
histogram_PC17, histogram_PC18,
             histogram_PC19, histogram_PC20, histogram_PC21, histogram_PC22,
histogram_PC23, histogram_PC24)

histogram_PC25 <- scaled_centered_pokemon %>%
  ggplot(aes(PC25)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC26 <- scaled_centered_pokemon %>%
  ggplot(aes(PC26)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC27 <- scaled_centered_pokemon %>%
  ggplot(aes(PC27)) +
```

```
  geom_histogram() +
  ylab("Frequency")

histogram_PC28 <- scaled_centered_pokemon %>%
  ggplot(aes(PC28)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC29 <- scaled_centered_pokemon %>%
  ggplot(aes(PC29)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC30 <- scaled_centered_pokemon %>%
  ggplot(aes(PC30)) +
  geom_histogram() +
  ylab("Frequency")

histogram_PC31 <- scaled_centered_pokemon %>%
  ggplot(aes(PC31)) +
  geom_histogram() +
  ylab("Frequency")

grid.arrange(histogram_PC25, histogram_PC26, histogram_PC27,
             histogram_PC28, histogram_PC29, histogram_PC30,
             histogram_PC31)

#boxplots before spatial sign
boxplot_PC1 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC1)) +
  coord_flip()

boxplot_PC2 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC2)) +
  coord_flip()

boxplot_PC3 <- scaled_centered_pokemon %>%
  ggplot() +
```

```
  geom_boxplot(aes(PC3)) +
  coord_flip()

boxplot_PC4 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC4)) +
  coord_flip()

boxplot_PC5 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC5)) +
  coord_flip()

boxplot_PC6 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC6)) +
  coord_flip()

boxplot_PC7 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC7)) +
  coord_flip()

boxplot_PC8 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC8)) +
  coord_flip()

boxplot_PC9 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC9)) +
  coord_flip()

boxplot_PC10 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC10)) +
  coord_flip()

boxplot_PC11 <- scaled_centered_pokemon %>%
  ggplot() +
```

```
  geom_boxplot(aes(PC11)) +
  coord_flip()

boxplot_PC12 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC12)) +
  coord_flip()

grid.arrange(boxplot_PC1, boxplot_PC2, boxplot_PC3, boxplot_PC4, boxplot_PC5,
boxplot_PC6,
             boxplot_PC7, boxplot_PC8, boxplot_PC9, boxplot_PC10, boxplot_PC11,
boxplot_PC12)

boxplot_PC13 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC13)) +
  coord_flip()

boxplot_PC14 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC14)) +
  coord_flip()

boxplot_PC15 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC15)) +
  coord_flip()

boxplot_PC16 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC16)) +
  coord_flip()

boxplot_PC17 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC17)) +
  coord_flip()

boxplot_PC18 <- scaled_centered_pokemon %>%
  ggplot() +
```

```
  geom_boxplot(aes(PC18)) +
  coord_flip()

boxplot_PC19 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC19)) +
  coord_flip()

boxplot_PC20 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC20)) +
  coord_flip()

boxplot_PC21 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC21)) +
  coord_flip()

boxplot_PC22 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC22)) +
  coord_flip()

boxplot_PC23 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC23)) +
  coord_flip()

boxplot_PC24 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC24)) +
  coord_flip()

grid.arrange(boxplot_PC13, boxplot_PC14, boxplot_PC15, boxplot_PC16, boxplot_PC17,
boxplot_PC18,
             boxplot_PC19, boxplot_PC20, boxplot_PC21, boxplot_PC22, boxplot_PC23,
boxplot_PC24)

boxplot_PC25 <- scaled_centered_pokemon %>%
  ggplot() +
```

```
  geom_boxplot(aes(PC25)) +
  coord_flip()

boxplot_PC26 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC26)) +
  coord_flip()

boxplot_PC27 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC27)) +
  coord_flip()

boxplot_PC28 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC28)) +
  coord_flip()

boxplot_PC29 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC29)) +
  coord_flip()

boxplot_PC30 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC30)) +
  coord_flip()

boxplot_PC31 <- scaled_centered_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC31)) +
  coord_flip()

grid.arrange(boxplot_PC25, boxplot_PC26, boxplot_PC27,
             boxplot_PC28, boxplot_PC29, boxplot_PC30, boxplot_PC31)

# Spatial Sign
spatial_sign <- spatialSign(scaled_centered_pokemon)
prepared_pokemon <- data.frame(spatial_sign)
# columns:31
```

```
# Boxplots after Spatial Sign
t_boxplot_PC1 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC1)) +
  coord_flip()

t_boxplot_PC2 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC2)) +
  coord_flip()

t_boxplot_PC3 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC3)) +
  coord_flip()

t_boxplot_PC4 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC4)) +
  coord_flip()

t_boxplot_PC5 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC5)) +
  coord_flip()

t_boxplot_PC6 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC6)) +
  coord_flip()

t_boxplot_PC7 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC7)) +
  coord_flip()

t_boxplot_PC8 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC8)) +
```

```
  coord_flip()

t_boxplot_PC9 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC9)) +
  coord_flip()

t_boxplot_PC10 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC10)) +
  coord_flip()

t_boxplot_PC11 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC11)) +
  coord_flip()

t_boxplot_PC12 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC12)) +
  coord_flip()

        grid.arrange(t_boxplot_PC1, t_boxplot_PC2, t_boxplot_PC3, t_boxplot_PC4,
t_boxplot_PC5, t_boxplot_PC6,
         t_boxplot_PC7, t_boxplot_PC8, t_boxplot_PC9, t_boxplot_PC10, t_boxplot_PC11,
t_boxplot_PC12)

t_boxplot_PC13 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC13)) +
  coord_flip()

t_boxplot_PC14 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC14)) +
  coord_flip()

t_boxplot_PC15 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC15)) +
```

```
  coord_flip()

t_boxplot_PC16 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC16)) +
  coord_flip()

t_boxplot_PC17 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC17)) +
  coord_flip()

t_boxplot_PC18 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC18)) +
  coord_flip()

t_boxplot_PC19 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC19)) +
  coord_flip()

t_boxplot_PC20 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC20)) +
  coord_flip()

t_boxplot_PC21 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC21)) +
  coord_flip()

t_boxplot_PC22 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC22)) +
  coord_flip()

t_boxplot_PC23 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC23)) +
```

```
  coord_flip()

t_boxplot_PC24 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC24)) +
  coord_flip()

grid.arrange(t_boxplot_PC13, t_boxplot_PC14, t_boxplot_PC15, t_boxplot_PC16,
t_boxplot_PC17, t_boxplot_PC18,
             t_boxplot_PC19, t_boxplot_PC20, t_boxplot_PC21, t_boxplot_PC22, t_boxplot_PC23,
t_boxplot_PC24)

t_boxplot_PC25 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC25)) +
  coord_flip()

t_boxplot_PC26 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC26)) +
  coord_flip()

t_boxplot_PC27 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC27)) +
  coord_flip()

t_boxplot_PC28 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC28)) +
  coord_flip()

t_boxplot_PC29 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC29)) +
  coord_flip()

t_boxplot_PC30 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC30)) +
```

```
  coord_flip()

t_boxplot_PC31 <- prepared_pokemon %>%
  ggplot() +
  geom_boxplot(aes(PC31)) +
  coord_flip()

grid.arrange(t_boxplot_PC25, t_boxplot_PC26, t_boxplot_PC27,
             t_boxplot_PC28, t_boxplot_PC29, t_boxplot_PC30, t_boxplot_PC31)


# Turning pokemon ID into pokemon names --------------------------------------

response_pokemon[response_pokemon == 1] <- "Bulbasaur"
response_pokemon[response_pokemon == 2] <- "Ivysaur"
response_pokemon[response_pokemon == 3] <- "Venusaur"
response_pokemon[response_pokemon == 4] <- "Charmander"
response_pokemon[response_pokemon == 5] <- "Charmeleon"
response_pokemon[response_pokemon == 6] <- "Charizard"
response_pokemon[response_pokemon == 7] <- "Squirtle"
response_pokemon[response_pokemon == 8] <- "Wartortle"
response_pokemon[response_pokemon == 9] <- "Blastoise"
response_pokemon[response_pokemon == 10] <- "Caterpie"
response_pokemon[response_pokemon == 11] <- "Metapod"
response_pokemon[response_pokemon == 12] <- "Butterfree"
response_pokemon[response_pokemon == 13] <- "Weedle"
response_pokemon[response_pokemon == 14] <- "Kakuna"
response_pokemon[response_pokemon == 15] <- "Beedrill"
response_pokemon[response_pokemon == 16] <- "Pidgey"
response_pokemon[response_pokemon == 17] <- "Pidgeotto"
response_pokemon[response_pokemon == 18] <- "Pidgeot"
response_pokemon[response_pokemon == 19] <- "Rattata"
response_pokemon[response_pokemon == 20] <- "Raticate"
response_pokemon[response_pokemon == 21] <- "Spearow"
response_pokemon[response_pokemon == 22] <- "Fearow"
response_pokemon[response_pokemon == 23] <- "Ekans"
response_pokemon[response_pokemon == 24] <- "Arbok"
response_pokemon[response_pokemon == 25] <- "Pikachu"
response_pokemon[response_pokemon == 26] <- "Raichu"
response_pokemon[response_pokemon == 27] <- "Sandshrew"
```

```
response_pokemon[response_pokemon == 28] <- "Sandslash"
response_pokemon[response_pokemon == 29] <- "NidoranF"
response_pokemon[response_pokemon == 30] <- "Nidorina"
response_pokemon[response_pokemon == 31] <- "Nidoqueen"
response_pokemon[response_pokemon == 32] <- "NidoranM"
response_pokemon[response_pokemon == 33] <- "Nidorino"
response_pokemon[response_pokemon == 34] <- "Nidoking"
response_pokemon[response_pokemon == 35] <- "Clefairy"
response_pokemon[response_pokemon == 36] <- "Clefable"
response_pokemon[response_pokemon == 37] <- "Vulpix"
response_pokemon[response_pokemon == 38] <- "Ninetales"
response_pokemon[response_pokemon == 39] <- "Jigglypuff"
response_pokemon[response_pokemon == 40] <- "Wigglytuff"
response_pokemon[response_pokemon == 41] <- "Zubat"
response_pokemon[response_pokemon == 42] <- "Golbat"
response_pokemon[response_pokemon == 43] <- "Oddish"
response_pokemon[response_pokemon == 44] <- "Gloom"
response_pokemon[response_pokemon == 45] <- "Vileplume"
response_pokemon[response_pokemon == 46] <- "Paras"
response_pokemon[response_pokemon == 47] <- "Parasect"
response_pokemon[response_pokemon == 48] <- "Venonat"
response_pokemon[response_pokemon == 49] <- "Venomoth"
response_pokemon[response_pokemon == 50] <- "Diglett"
response_pokemon[response_pokemon == 51] <- "Dugtrio"
response_pokemon[response_pokemon == 52] <- "Meowth"
response_pokemon[response_pokemon == 53] <- "Persian"
response_pokemon[response_pokemon == 54] <- "Psyduck"
response_pokemon[response_pokemon == 55] <- "Golduck"
response_pokemon[response_pokemon == 56] <- "Mankey"
response_pokemon[response_pokemon == 57] <- "Primeape"
response_pokemon[response_pokemon == 58] <- "Growlithe"
response_pokemon[response_pokemon == 59] <- "Arcanine"
response_pokemon[response_pokemon == 60] <- "Poliwag"
response_pokemon[response_pokemon == 61] <- "Poliwhirl"
response_pokemon[response_pokemon == 62] <- "Poliwrath"
response_pokemon[response_pokemon == 63] <- "Abra"
response_pokemon[response_pokemon == 64] <- "Kadabra"
response_pokemon[response_pokemon == 65] <- "Alakazam"
response_pokemon[response_pokemon == 66] <- "Machop"
response_pokemon[response_pokemon == 67] <- "Machoke"
```

```
response_pokemon[response_pokemon == 68] <- "Machamp"
response_pokemon[response_pokemon == 69] <- "Bellsprout"
response_pokemon[response_pokemon == 70] <- "Weepinbell"
response_pokemon[response_pokemon == 71] <- "Victreebel"
response_pokemon[response_pokemon == 72] <- "Tentacool"
response_pokemon[response_pokemon == 73] <- "Tantacruel"
response_pokemon[response_pokemon == 74] <- "Geodude"
response_pokemon[response_pokemon == 75] <- "Graveler"
response_pokemon[response_pokemon == 76] <- "Golem"
response_pokemon[response_pokemon == 77] <- "Ponyta"
response_pokemon[response_pokemon == 78] <- "Rapidash"
response_pokemon[response_pokemon == 79] <- "Slowpoke"
response_pokemon[response_pokemon == 80] <- "Slowbro"
response_pokemon[response_pokemon == 81] <- "Magnemite"
response_pokemon[response_pokemon == 82] <- "Magnetron"
response_pokemon[response_pokemon == 83] <- "Farfetch'd"
response_pokemon[response_pokemon == 84] <- "Doduo"
response_pokemon[response_pokemon == 85] <- "Dodio"
response_pokemon[response_pokemon == 86] <- "Seel"
response_pokemon[response_pokemon == 87] <- "Dewgong"
response_pokemon[response_pokemon == 88] <- "Grimer"
response_pokemon[response_pokemon == 89] <- "Muk"
response_pokemon[response_pokemon == 90] <- "Shellder"
response_pokemon[response_pokemon == 91] <- "Cloyster"
response_pokemon[response_pokemon == 92] <- "Gastly"
response_pokemon[response_pokemon == 93] <- "Haunter"
response_pokemon[response_pokemon == 94] <- "Gengar"
response_pokemon[response_pokemon == 95] <- "Onix"
response_pokemon[response_pokemon == 96] <- "Drowzee"
response_pokemon[response_pokemon == 97] <- "Hypno"
response_pokemon[response_pokemon == 98] <- "Krabby"
response_pokemon[response_pokemon == 99] <- "Kingler"
response_pokemon[response_pokemon == 100] <- "Voltorb"
response_pokemon[response_pokemon == 101] <- "Electrode"
response_pokemon[response_pokemon == 102] <- "Exeggcute"
response_pokemon[response_pokemon == 103] <- "Exeggutor"
response_pokemon[response_pokemon == 104] <- "Cubone"
response_pokemon[response_pokemon == 105] <- "Marowak"
response_pokemon[response_pokemon == 106] <- "Hitmonlee"
response_pokemon[response_pokemon == 107] <- "Hitmonchan"
```

```
response_pokemon[response_pokemon == 108] <- "Lickitung"
response_pokemon[response_pokemon == 109] <- "Koffing"
response_pokemon[response_pokemon == 110] <- "Weezing"
response_pokemon[response_pokemon == 111] <- "Rhyhorn"
response_pokemon[response_pokemon == 112] <- "Rhydon"
response_pokemon[response_pokemon == 113] <- "Chansey"
response_pokemon[response_pokemon == 114] <- "Tangela"
response_pokemon[response_pokemon == 115] <- "Kangaskhan"
response_pokemon[response_pokemon == 116] <- "Horsea"
response_pokemon[response_pokemon == 117] <- "Seadra"
response_pokemon[response_pokemon == 118] <- "Goldeen"
response_pokemon[response_pokemon == 119] <- "Seaking"
response_pokemon[response_pokemon == 120] <- "Staryu"
response_pokemon[response_pokemon == 121] <- "Starmie"
response_pokemon[response_pokemon == 122] <- "MrMime"
response_pokemon[response_pokemon == 123] <- "Scyther"
response_pokemon[response_pokemon == 124] <- "Jynx"
response_pokemon[response_pokemon == 125] <- "Electabuzz"
response_pokemon[response_pokemon == 126] <- "Magmar"
response_pokemon[response_pokemon == 127] <- "Pinsir"
response_pokemon[response_pokemon == 128] <- "Tauros"
response_pokemon[response_pokemon == 129] <- "Magikarp"
response_pokemon[response_pokemon == 130] <- "Gyarados"
response_pokemon[response_pokemon == 131] <- "Lapras"
response_pokemon[response_pokemon == 132] <- "Ditto"
response_pokemon[response_pokemon == 133] <- "Eevee"
response_pokemon[response_pokemon == 134] <- "Vaporeon"
response_pokemon[response_pokemon == 135] <- "Jolteon"
response_pokemon[response_pokemon == 136] <- "Flareon"
response_pokemon[response_pokemon == 137] <- "Porygon"
response_pokemon[response_pokemon == 138] <- "Omanyte"
response_pokemon[response_pokemon == 139] <- "Omastar"
response_pokemon[response_pokemon == 140] <- "Kabuto"
response_pokemon[response_pokemon == 141] <- "Kabutops"
response_pokemon[response_pokemon == 142] <- "Aerodactyl"
response_pokemon[response_pokemon == 143] <- "Snorlax"
response_pokemon[response_pokemon == 144] <- "Articuno"
response_pokemon[response_pokemon == 145] <- "Zapdos"
response_pokemon[response_pokemon == 146] <- "Moltres"
response_pokemon[response_pokemon == 147] <- "Dratini"
```

```
response_pokemon[response_pokemon == 148] <- "Dragonair"
response_pokemon[response_pokemon == 149] <- "Dragonite"
response_pokemon[response_pokemon == 150] <- "Mewtwo"
response_pokemon[response_pokemon == 151] <- "Mew"

# Data Splitting ------------------------------------------------------------

# Check Response Balance
ggplot(data = response_pokemon) +
  geom_bar(aes(pokemonId)) +
  labs(x = "Pokemon ID", y = "Frequency", title = "Distribution of Pokemon ID")

# Data Splitting using Stratified Random Sampling
set.seed(1234)

subset_rows <- createDataPartition(response_pokemon$pokemonId, p = .25, list = FALSE)
response_subset <- response_pokemon[subset_rows, ]
pokemon_subset <- prepared_pokemon[subset_rows, ]

response_subset <- as.data.frame(response_subset)
colnames(response_subset) <- "pokemonId"

training_rows <- createDataPartition(response_subset$pokemonId, p = .80, list = FALSE)

training_predictors <- pokemon_subset[training_rows, ] # obs: 236786 columns: 31
training_response <- response_subset[training_rows, ] # obs: 23676 columns: 1

testing_predictors <- pokemon_subset[-training_rows, ] # obs: 59196 columns: 31
testing_response <- response_subset[-training_rows, ] # obs: 59196 columns: 1

# removing observations that only occur once
combined_training <- data.frame(training_response, training_predictors)

training_single_observations <- combined_training %>%
  group_by(training_response) %>%
  tally() %>%
  filter(n == 1) %>%
  select(-n)

c <- as.character(training_single_observations$training_response)
```

```
reduced_training <- combined_training[!combined_training$training_response %in% c, ]

reduced_training_response <- reduced_training$training_response
reduced_training_predictors <- reduced_training[, 2:32]

# making factors
training_response <- as.factor(training_response)
reduced_training_response <- as.factor(reduced_training_response)
testing_response <- as.factor(testing_response)

# Linear Classification Models -----------------------------------------------

# Logistic Regression WORKS ON A TEENY SAMPLE SO IT'LL TAKE TIME
ctrl <- trainControl(
  method = "LGOCV",
  summaryFunction = defaultSummary,
  savePredictions = TRUE
)

set.seed(1234)
logistic_regression <- train(training_predictors,
                             y = training_response,
                             method = "multinom",
                             metric = "Kappa",
                             trControl = ctrl,
                             maxit = 5,
                             MaxNWts = 4896
)

logistic_regression

plot(logistic_regression, main = "Plot of Logistic Regression")

confusionMatrix(
  data = logistic_regression$pred$pred,
  reference = logistic_regression$pred$obs
)

# Linear Discriminant Analysis
```

```
set.seed(1234)
lda_model <- train(training_predictors,
                   y = training_response,
                   method = "lda",
                   metric = "Kappa",
                   trControl = ctrl
)

lda_model

plot(lda_model, main = "Plot of Linear Discriminant Analysis")

confusionMatrix(
  data = lda_model$pred$pred,
  reference = lda_model$pred$obs
)

# Partial Least Squares Discriminant Analysis WORKS ON A TEENY SAMPLE SO IT'LL
TAKE TIME
ctrl <- trainControl(
  summaryFunction = defaultSummary,
  savePredictions = TRUE
)

set.seed(1234)
pls_model <- train(
  x = training_predictors,
  y = training_response,
  method = "pls",
  tuneGrid = expand.grid(.ncomp = 1:20),
  metric = "Kappa",
  trControl = ctrl,
  maxit = 10000
)

pls_model

plot(pls_model, main = "Plot of PLS Discriminant Analysis")

confusionMatrix(
```

```
  data = pls_model$pred$pred,
  reference = pls_model$pred$obs
)

# Penalized Model ERROR: NO CLASS CAN HAVE 1 OR 0 OBSERVATIONS
ctrl <- trainControl(
  method = "LGOCV",
  summaryFunction = defaultSummary,
  savePredictions = TRUE
)

glmnGrid <- expand.grid(
  .alpha = c(0, .1, .2, .4, .6, .8, 1),
  .lambda = seq(.01, .2, length = 10)
)

set.seed(123)
penalized_model <- train(
  x = reduced_training_predictors,
  y = reduced_training_response,
  method = "glmnet",
  tuneGrid = glmnGrid,
  metric = "Kappa",
  trControl = ctrl
)

penalized_model

plot(penalized_model, main = "Plot of Penalized Model")

confusionMatrix(
  data = penalized_model$pred$pred,
  reference = penalized_model$pred$obs
)

# Non-Linear Classification Models -------------------------------------------

# Quadratic Regularized Discriminant Analysis NEEDS TO REMOVE SINGLE RECORD
CLASSES
ctrl_nonLinear_models <- trainControl(
```

```
  method = "LGOCV",
  number = 10,
  classProbs = FALSE,
  savePredictions = TRUE,
  summaryFunction = defaultSummary
)
set.seed(123)
qda_model <- train(
  x = reduced_training_predictors,
  y = reduced_training_response,
  method = "qda",
  metric = "Kappa",
  trControl = ctrl_nonLinear_models
)

qda_model

plot(qda_model, main = "Plot of Quadratic Regularized Discriminant Analysis")

confusionMatrix(
  data = qda_model$pred$pred,
  reference = qda_model$pred$obs
)

# Regularized Discriminant Analysis RETURNING ONLY ZEROS FOR ACCURACY AND
KAPPA
set.seed(123)
rda_model <- train(
  x = training_predictors,
  y = training_response,
  method = "rda",
  metric = "Kappa",
  tuneGrid = expand.grid(.lambda = 1:3, .gamma = 1:3),
  trControl = ctrl_nonLinear_models
)

rda_model

plot(rda_model, main = "Plot of Regularized Discriminant Analysis")
```

```
confusionMatrix(
  data = rda_model$pred$pred,
  reference = rda_model$pred$obs
)

# Mixture Discriminant Analysis
set.seed(123)
mda_model <- train(
  x = training_predictors,
  y = training_response,
  method = "mda",
  metric = "Kappa",
  tuneGrid = expand.grid(.subclasses = 1:10),
  trControl = ctrl_nonLinear_models
)

mda_model

plot(mda_model, main = "Plot of Mixture Discriminant Analysis")

confusionMatrix(
  data = mda_model$pred$pred,
  reference = mda_model$pred$obs
)

# Neural Networks
ctrl_nonLinear_models <- trainControl(
  method = "LGOCV",
  number = 10,
  classProbs = FALSE,
  savePredictions = TRUE,
  summaryFunction = defaultSummary
)
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (31 + 1) + (maxSize + 1) * 144) ## 31 is the number of predictors, 144
classes (pokemon IDs)

nnet_model <- train(
  x = training_predictors,
```

```
  y = training_response,
  method = "nnet",
  metric = "Kappa",
  tuneGrid = nnetGrid,
  trace = FALSE,
  maxit = 100,
  MaxNWts = numWts,
  trControl = ctrl_nonLinear_models
)

nnet_model

plot(nnet_model, main = "Plot of Neural Networks")

confusionMatrix(
  data = nnet_model$pred$pred,
  reference = nnet_model$pred$obs
)


# Flexible Discriminant Analysis
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:35)

set.seed(1234)
fda_model <- train(
  x = training_predictors,
  y = training_response,
  method = "fda",
  tuneGrid = marsGrid,
  trControl = ctrl_nonLinear_models
)

fda_model

plot(fda_model, main = "Plot of Flexible Discriminant Analysis")

confusionMatrix(
  data = fda_model$pred$pred,
  reference = fda_model$pred$obs
)
```

```
# Support Vector Machines
sigmaRangeReduced <- sigest(as.matrix(training_predictors))
svmRGridReduced <- expand.grid(
  .sigma = sigmaRangeReduced[1],
  .C = 2^(seq(-4, 6))
)
set.seed(123)
svm_model <- train(
  x = training_predictors,
  y = training_response,
  method = "svmRadial",
  metric = "Kappa",
  tuneGrid = svmRGridReduced,
  fit = FALSE,
  trainControl = ctrl_nonLinear_models
)

svm_model

plot(svm_model, main = "Plot of Support Vector Machine")

confusionMatrix(
  data = svm_model$pred$pred,
  reference = svm_model$pred$obs
)

# K-Nearest Neighbors
set.seed(123)
knn_model <- train(
  x = training_predictors,
  y = training_response,
  method = "knn",
  metric = "Kappa",
  ## tuneGrid = data.frame(.k = c(4*(0:5)+1, 20*(1:5)+1, 50*(2:9)+1)), ## 21 is the best
  tuneGrid = data.frame(.k = 1:50),
  trControl = ctrl_nonLinear_models
)

knn_model
```

```
plot(knn_model, main = "Plot of K-Nearest Neighbors")

confusionMatrix(
  data = knn_model$pred$pred,
  reference = knn_model$pred$obs
)

# Naive Bayes NEED TO REMOVE SINGLE OBSERVATIONS
set.seed(123)
nb_model <- train(
  x = reduced_training_predictors,
  y = reduced_training_response,
  method = "nb",
  metric = "Kappa",
  tuneGrid = data.frame(.fL = 2, .usekernel = TRUE, .adjust = TRUE),
  trControl = ctrl_nonLinear_models
)

nb_model

plot(nb_model, main = "Plot of Naive Bayes")

confusionMatrix(
  data = nb_model$pred$pred,
  reference = nb_model$pred$obs
)

# Predictions -----------------------------------------------------

# predicting on testing side for two best models
pred1 <- predict(knn_model, newdata = testing_predictors)
postResample(pred = pred1, obs = testing_response)

pred2 <- predict(nb_model, obs = testing_predictors)
postResample(pred = pred2, obs = testing_response)

# confusion matrix for two best models
confusionMatrix(
  data = knn_model$pred$pred,
```

```
  reference = knn_model$pred$obs
)

confusionMatrix(
  data = nb_model$pred$pred,
  reference = nb_model$pred$obs
)

# variance importance
imp1 <- filterVarImp(knn_model, estimate = "Kappa", scale = FALSE)
plot(imp1, top = 5, main = "K-nearest Neighbor")

imp2 <- varImp(nb_model, scale = FALSE)
plot(imp2, top = 5, main = "Naive Bayes Model")
```