Part 4: Final Report

Alex Gui and Lathan Liou

Date Submitted:

Introduction

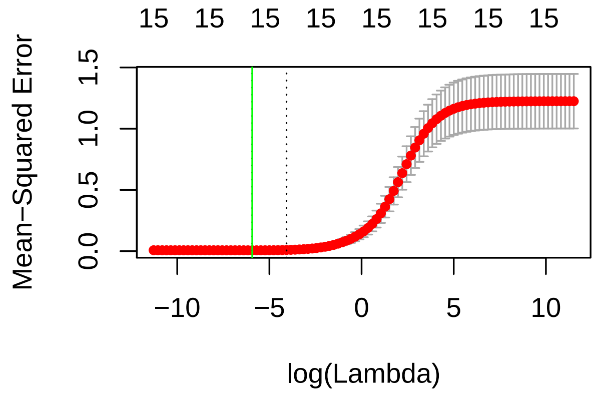
In July of 2016, Pokemon Go became an overnight sensation with hundreds of millions of people having downloaded the mobile game. The whole point of the game is to try to catch all the Pokemon available and train them (increase their combat power, which is abbreviated cp) so that you can battle other players with your strengthened Pokemon. A quick note about Pokemon is that they can evolve into stronger forms, so an evolved Pokemon will generally always have a higher cp than a non-evolved Pokemon. A number of people have tried to generate models in an attempt to predict the best way to maximize cp for their Pokemon. This is what we will attempt to do ourselves: create a model to predict combat power for an evolved Pokemon.

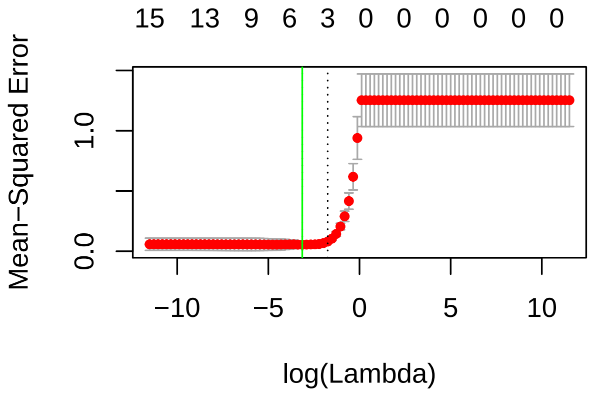
To refresh your memory, the dataset we are looking at is an original data set collected by *OpenIntro*[1], most likely by some individual who was playing Pokemon Go and decided to record data. The dataset contains 75 observations across 26 variables, with each observation representing a randomly generated Pokemon that the gamer caught. Four species are represented in this data, so the conclusions drawn from this modeling process will reflect the population of these 4 particular species: Eevee, Pidgey, Caterpie, and Weedle.

We avoid using "new" predictor variables in our modeling process because as a user, you wouldn't have access to any of the "new" information, but if you're interested in whether your pokemon will evolve into one with a high cp (*cp_new*), you would want to know which of the Pokemon's current stats can indicate a high *cp_new*. The variables that we are particularly interested in are cp, hp, and power_up_stardust. Our intuition is that a pokemon with a higher cp might evolve into a pokemon with a higher cp. Hp, or hit points, refers to the amount of damage a Pokemon can sustain in battle before fainting. It would be interesting to see whether a Pokemon with high hp will also have high cp. Power up stardust is used to raise cp of the pokemon, but the catch here is we don't know the ideal amount of power up stardust to max out the *cp_new* of the evolved pokemon.

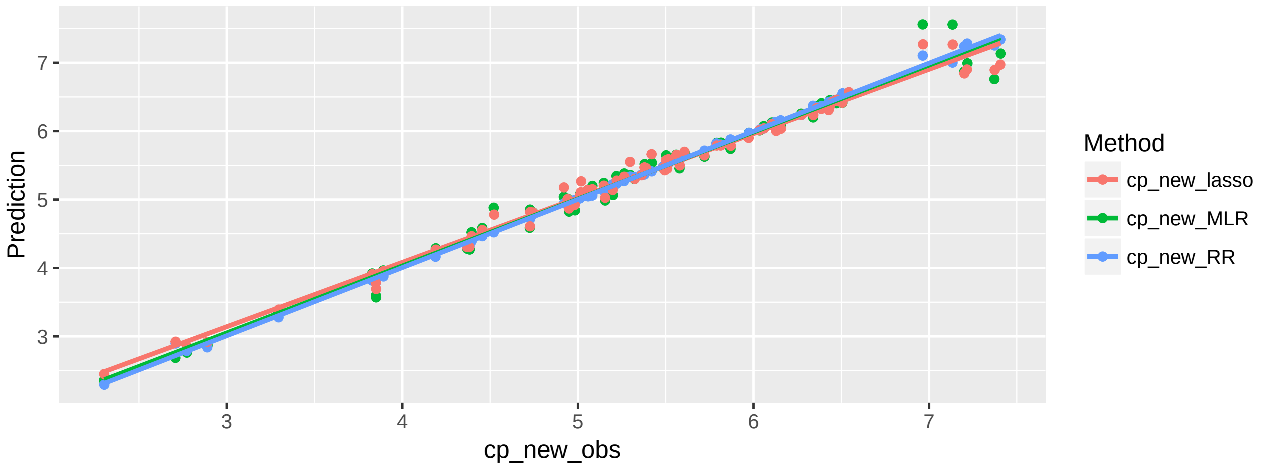
You can follow our work here: https://github.com/alexaaag/math158-project.

Ridge Regression and Lasso



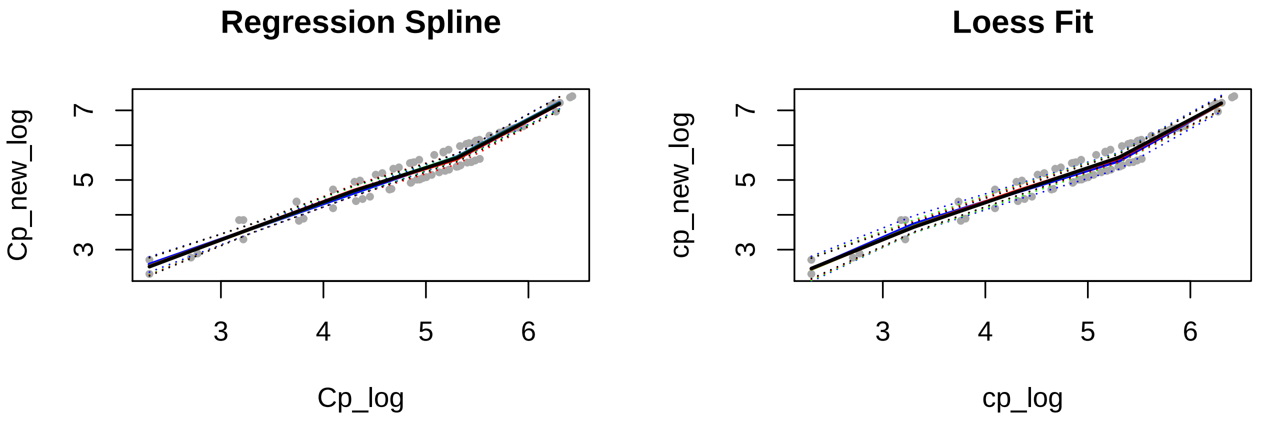


Ridge regression shrunk the coefficients closer to zero than multiple linear regression did. For instance, if we compare the coefficients of cp_{log}, it is 1.186 in our linear regression model, and 1.24e-5 in the ridge regression model. On the other hand, lasso regression not only shrunk the coefficients but also performed variable selection, selecting cp_{log}, $attack\_strong\_value$ and $attack\_weak\_value$ to name a few.



From this figure, it seems ridge regression and lasso seem to predict very similarly as multiple linear regression, since the slopes of each regression fit basically overlap each other.

Smoothing



Both the smoothing spline and the loess curves fit the data extremely well. Changing the degrees of freedom, and hence the number of knots, for the smoothing splines improves the fit minimally to a certain point past which increasing the degrees of freedom actually begins to increase SSE. Likewise, increasing the span from 0.2 for loess actually increases SSE.

I would choose the regression spline model with 5 degrees of freedom because it fits the data very smoothly, and it still has a functional form which lends itself to interpretability.

Conclusion

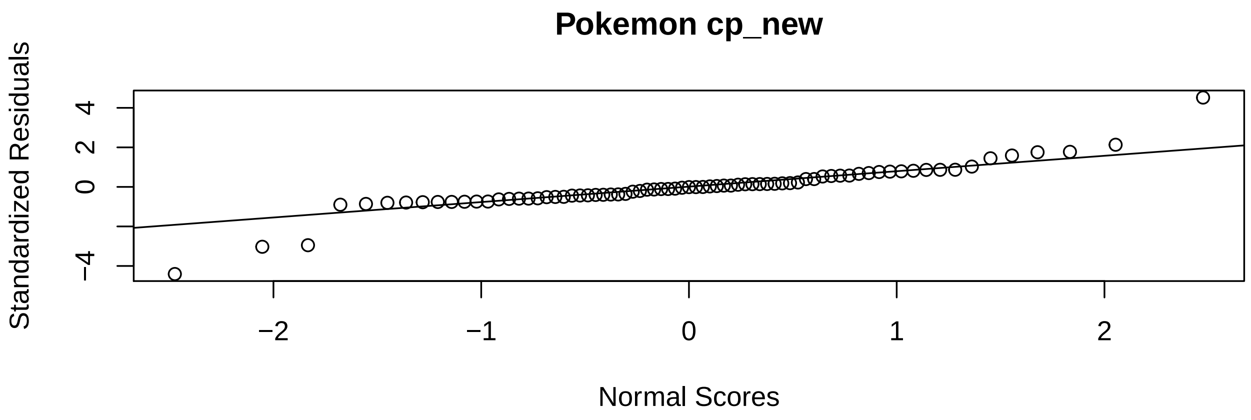
Overall, running ridge regression, lasso, and smoothing methods such as regresion splines and loess did not improve the model fit relative to multiple linear regression by very much. This is fairly unsurprising to us because we noticed how extremely well linear regression fit our dataset previously, and this can most likely be attributed to the nature of how cp_{new} is actually modeled in the game. We think cp_{new} is likely coded into the game as a function of a linear combination of certain predictors and our multiple linear regression model fairly closely matches the real model used in-game.

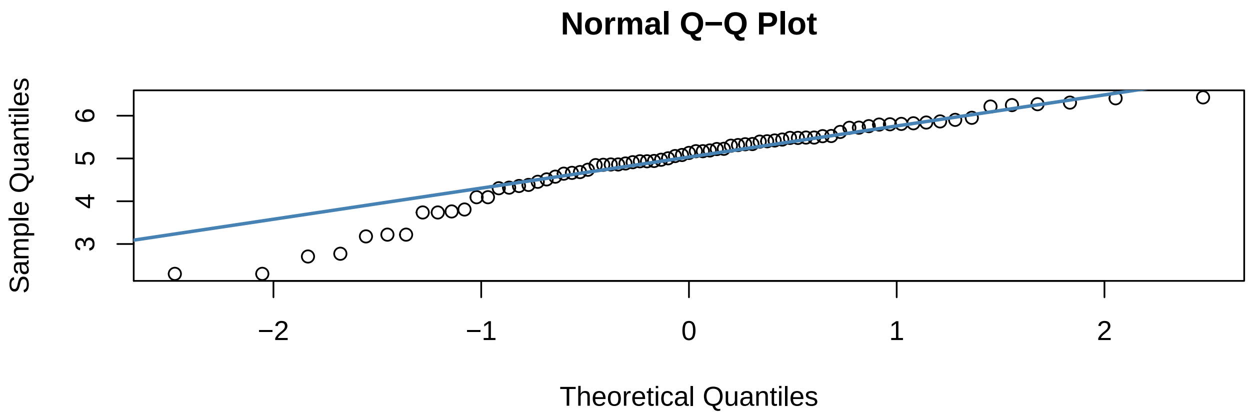
%Other questions???

Something New

Q-Q Plot

A normal probability plot is used to identify substantial departures from normality in the data. We chose in particular to plot what is known as a normal quantile-quantile plot (Q-Q plot in short), which plots sample quantiles against theoretical quantiles from a continuous cumultaive distribution function such as the standard normal distribution. Mathematically, the Q–Q plot draws the q-th quantile of theoretical cumulative probability distribution function F against the q-th quantile of our sample data for a range of values of q. Thus, the Q–Q plot is a parametric curve indexed over [0,1] with values in the real plane R^2. The units of a Q-Q plot are rankits, which are the expected values of the order statistics of a sample from the standard normal distribution the same size as the data. A $y = x$ reference line is also plotted and if the sample data also come from a normal distribution, the points should fall roughly along this reference line. A q-q plot is important because it can provide more information about the nature of the departure from normality and assess "goodness-of-fit" graphically.

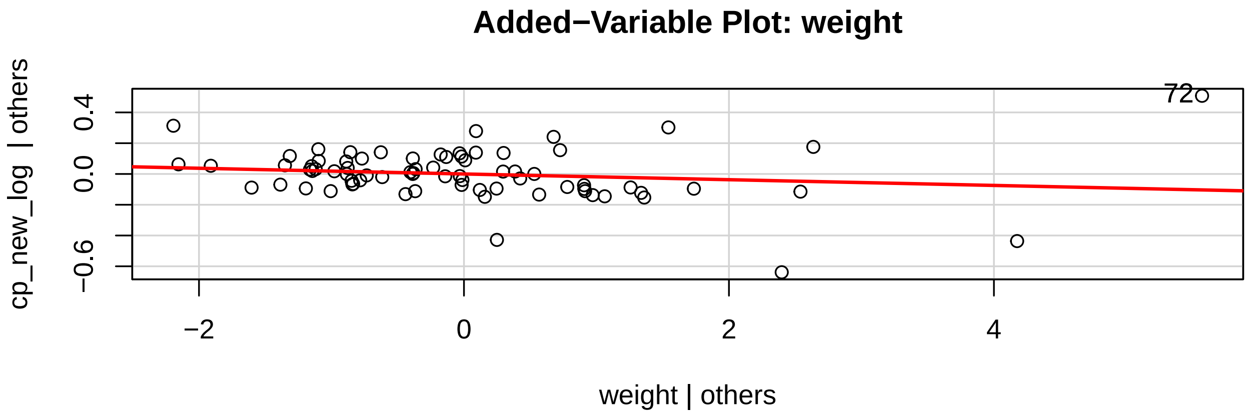




In our Q-Q plot, we see that the points do not fall on the line as they follow a nonlinear pattern, suggesting that *cp_log* does not follow a normal distribution. This is interesting to us because we see that even after we log-transformed cp, we see that there are still departures from normality, most likely caused by the Eevee outliers.

Added Variable Plots

Added variable plots, also known as partial regression plot, attempt to show the marginal effect of adding another variable to a model already having one or more independent variables. Added variable plots are formed by first computing the residuals of regressing the response variable against the independent variable(s) but omitting the variable of interest, X_i. Let's call this $Y._{[i]}$ Next, the residuals are computed from regressing X_i against the remaining independent variables. Let's call this $X_{i.[i]}$ The residuals from $Y._{[i]}$ and $X_{i.[i]}$ are then plotted against each other. For this analysis, the underlying assumption is that variables aren't highly correlated.



This plot contains the least squares line which has slope of -0.01861, which suggests that the linear relationship between *cp_new_log* and weight is slightly negative. The scatter of the points around the least square lines is about the same to the scatter around the horizontal line $e(cp\_new\_log|other\ variables)$, indicating that adding weight to the regression model does not substantially reduce the error sum of squares. In fact, the coefficient of partial determination for the linear effect of weight is $R^2_{Y weight|cp,species,attack\_strong\_value,hp} = 0.0283$.

Another benefit of an added variable plot is it allows us to determine influential points, after accounting for the other variables in the model.

Unsupervised Learning

In the previous write-ups, our analysis concerns predictions: given a set of explanatory variables $X_1, ...X_{p-1}$ and n observations, how can we predict the response variable Y? We have adopted various regression techniques and yielded satisfying results. However, in this part of analysis, we want to shift our focus away from the Y variable and ask ourselves: what are something interesting things we can say about the Xs? What are some underlying structures and how can we present them efficiently? We introduce the concept of unsupervised learning, "a set of statistical tools intended for the setting in which we have only a set of features X1, X2, . . . , Xp measured on n observations."

Principal Components Analysis(PCA)

Principal components analysis is widely used as an unsupervised learning method for feature extraction and data compression. In our analysis, we will apply principal components in our regression model as a dimensionality reduction technique. The intuition behind PCA is: given a set of highly correlated predictors, PCA will transform it into a smaller set of linearly independent variables called principal components. The transformation is defined such that the first principal component direction captures the greatest possible variability in the data, in others words, explain the most variability of the data. The succeeding principal components are linear combinations of the variables that is un- correlated with the preceding component and has largest variance subject to this constraint. The set of components constitute a basis for our data space.

Principal Components Regression

The principal components regression approach will first construct M principal components and then regress on the components instead of individual predictors. The underlying assumption of the model is "the directions in which $X_1, ...X_p$ shows the greatest variance are those associated with Y". Although this assumption is not guaranteed, it regardless provides a decent approximation that often yields good results. M, the number of principal components, is our tuning parameter that will be chosen by cross-validation.

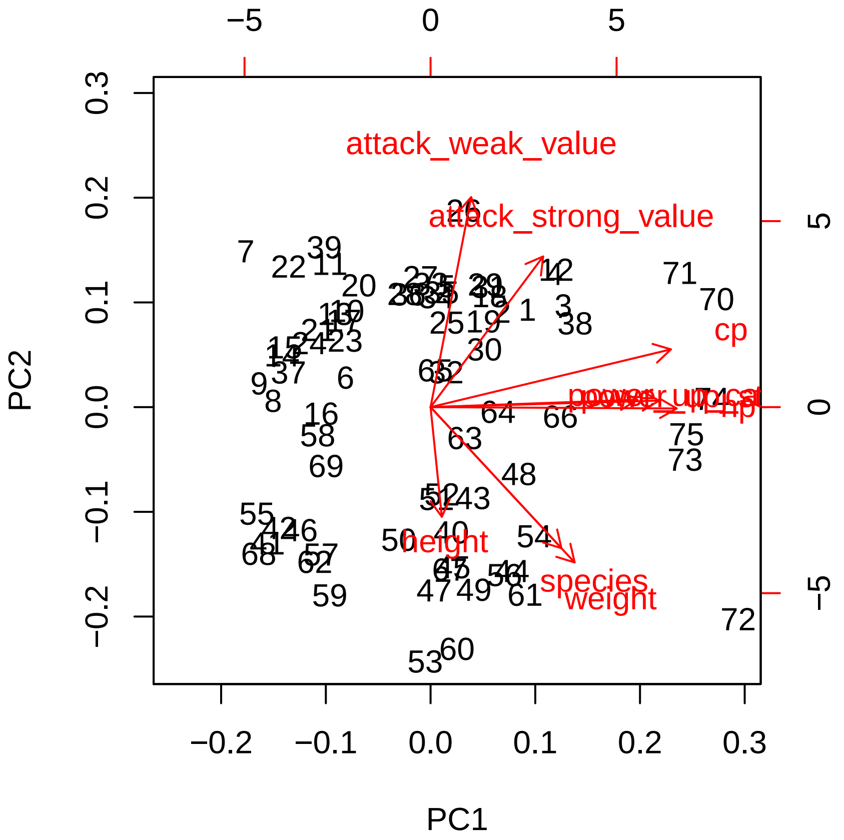
We believe PCA works well with our pokemon data given the existence of strong correlation among our predictors. Our analysis shows that PCR greatly reduced variance and contributed to our model's predictive power.

Principal Components

Our model first constructs 9 principal components(this makes sense since $p = 9$ and $M \leq p$).

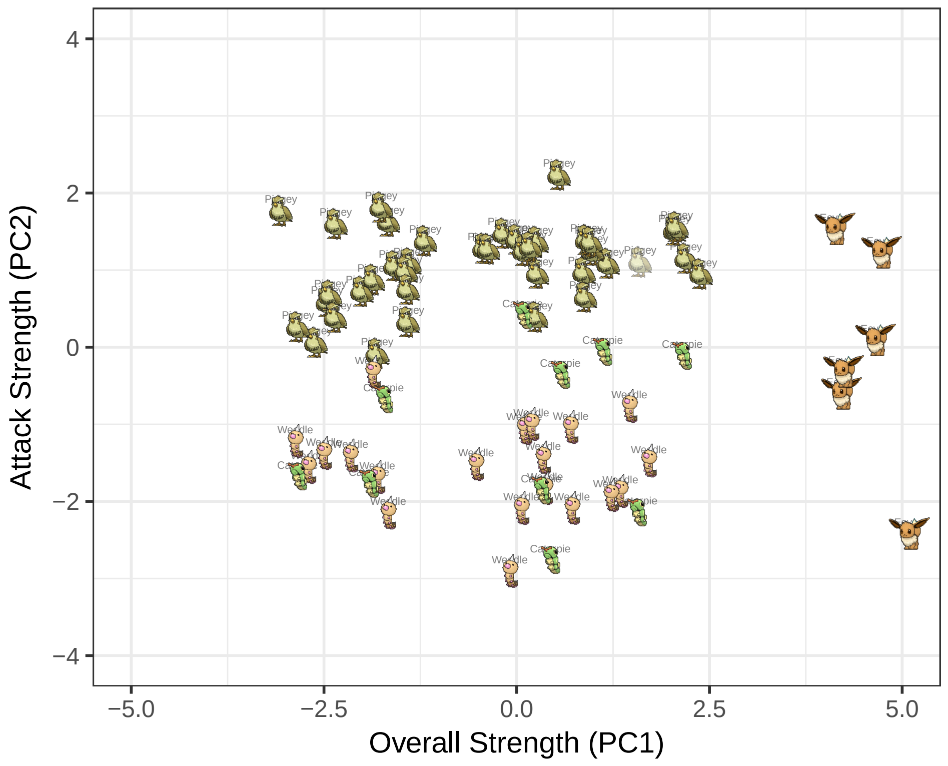
```
## Standard deviations:
## [1] 2.000 1.379 1.164 1.023 0.581 0.417 0.304 0.243 0.191
##
## Rotation:
##                        PC1     PC2    PC3      PC4     PC5     PC6     PC7
## species             0.2539 -0.3970  0.238 -0.52512  0.3707 -0.2008 -0.4838
## cp                  0.4664  0.1625  0.119 -0.00223 -0.1473  0.3512  0.1679
## hp                  0.4776 -0.0038 -0.106  0.03351 -0.0244  0.4642 -0.2838
## weight              0.2795 -0.4370  0.453  0.01056  0.0396 -0.0293  0.6677
## height              0.0219 -0.3086  0.287  0.81195  0.0283 -0.0865 -0.3673
## power_up_stardust   0.4434  0.0184 -0.354  0.12697  0.0397  0.0742 -0.0423
## power_up_candy      0.4011  0.0209 -0.426  0.12977  0.0511 -0.6973  0.1514
## attack_weak_value   0.0787  0.5910  0.286  0.14366  0.7268 -0.0244  0.0783
## attack_strong_value 0.2185  0.4239  0.491 -0.10070 -0.5527 -0.3466 -0.2137
##                         PC8     PC9
## species             -0.1821  0.0209
## cp                  -0.6273  0.4185
## hp                   0.6523  0.1931
## weight               0.2265 -0.1601
## height              -0.1271  0.0580
## power_up_stardust   -0.2290 -0.7748
## power_up_candy       0.0918  0.3465
## attack_weak_value    0.0754 -0.0396
## attack_strong_value  0.1171 -0.1902
```

To visualize it:



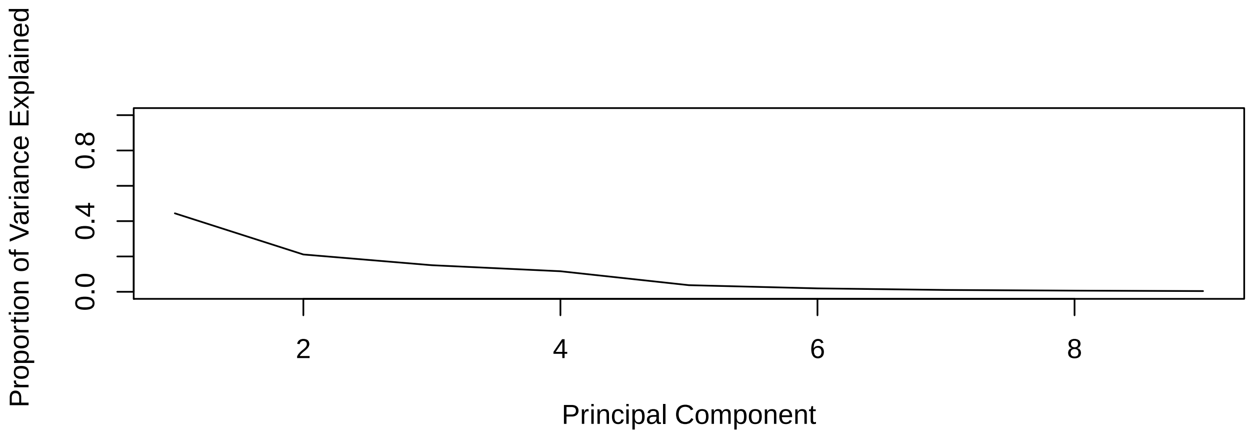
Look at PC1. We observe that the higher the performance metrics, the higher the PC1 value. Therefore we can interpret PC1 as a measurement of overall strength. As for PC2, we notice that higher PC2 is associated with higher attack value. Therefore we interpret PC2 as a measurement of attack strength.

We then plot our pokemon on our new principal components space:



Interesting Insights: Using principal components, we get to create two new powerful metrics to evaluate our pokemons. From the plot, you can observe that Eeeve in general has high overall strength and high attack strength. Pidgey has good attack stats but is weaker in general due to the creature's intrinsic nature. Caterpie and Weedle are weak on both metrics. Overall, this PCA gives you a high-level overview of our pokemon's strength. You should feel excited!

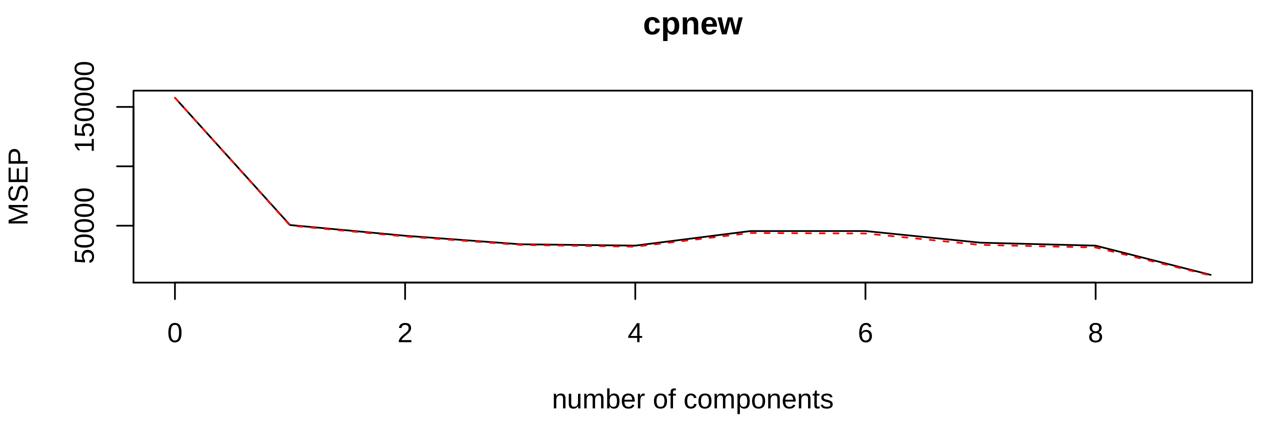
Regression:



Observe that the first two principal components explain nearly 60% of the variability in the data. As $M \to 10$, the marginal contribution to variability explained decreases. Our regression model will use cross validation to tune M, the number of components as predictors.

```
## Data:    X dimension: 50 9
##  Y dimension: 50 1
## Fit method: svdpc
## Number of components considered: 9
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           397.1    224.9    203.8    185.4    182.3    213.3    213.4
## adjCV        397.1    223.8    202.3    184.0    179.9    209.4    208.5
##        7 comps  8 comps  9 comps
## CV         189    182.3    92.81
## adjCV      184    177.9    89.94
##
## TRAINING: % variance explained
##        1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X        45.77    65.32    80.62    91.87    96.18    98.04    98.98
## cpnew    70.69    79.25    84.37    87.68    88.05    90.30    93.22
##        8 comps  9 comps
## X        99.56   100.00
## cpnew    94.48    98.72
```

The metric of cross-validation is *root_mean_squared_error*. Ideally, we want to pick M where the CV score is minimized.



We compute the test MSE as follows using $M = 9$ components:

```
pcr.pred = predict(pcr.fit, pokemondata[-train, ], ncomp = 9)
mean((pcr.pred - pokemondata[-train, ]$cpnew)^2)
```

```
## [1] 2827
```

Interpretation

Summary Report