APA Project

Pokemons

Enrique Reyes Illescas i Guillem Gili i Bueno
APA
Q1-2018/19
11/01/2019

INDEX

1. Introduction

The dataset has been obtained from information regarding the video-game series Pokemon. The dataset can be found at kaggle[1], and contains various characteristics identifying each pokemon, some related to combat, some related to other aspects of the game and other descriptive variables. Our goal is to predict the "*speed*" a pokemon will have given the rest of its characteristics, so we are trying to solve a regression problem.

This is a harder problem than it looks like, while the physics in the pokemon universe work similarly to the real world pokemons are magical creatures break them. What we want is to explore if the attributes make some sense and we can find patterns in this universe. We have the breeding compatibilities, the physical *shape*, abilities and the elemental resistances a pokemon has, besides the rest of their stats. What we want to know is whether those attributes and their battle attributes allow us to predict how mobile (their "*speed*") a pokemon would be.

2. Previous work and results

Other kaggle users have done some statistical analysis on the dataset[2]. Various stats have been plotted compared to *types*, had its density functions plottes,etc. We looked at some of the code to made plots that reflected the things we wanted to see and adapted thing to our needs.

We also looked at the information that could be obtained from the graphs to see how much of it was worth including in the project, but most of the plots were using just "*type_1*"(although "*type_2*" is equally important). They had also some other rigor errors that we found relevant. Due to this, we decided that it would be better to generate them ourselves and focus on what we thought was more relevant (pokemons that have a single type vs 2, "*weight*" and "*height*" distribution,etc).

3. Data visualization and transformation

3.1 Attribute description

We won't explain all the attributes in the dataset for simplicity. This is due to many being redundant or completely uninformative. These are attributes from the dataset we will be using in our regression:

- "*height*","*weight*","*shape*"<- Pokemon physical appearance attributes (not battle stats). Height and Weight are numerical
- "*base_experience*"<- Multiplier used to calculate how much experience a pokemon needs to level up. (not a battle stat)

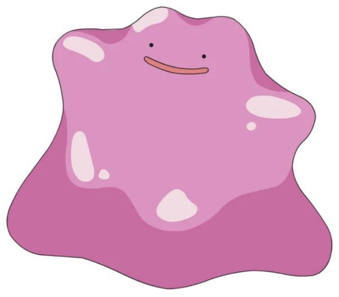
- "*type_1*","*type_2*"<- Types that defines a pokemon's elemental weakness and resistances.
- "*attack","defense","hp","special_attack","special_defense","speed*"<- Various stat base values for the pokemon(battle stat). Speed is our target variable to predict.
- "*ability_1","ability_2","ability_hidden*"<- Base ability the pokemon has. Abilities affect a variety of game aspects and may affect battles. A pokemon may have only 1 ability. Abilities 1 and 2 are equally likely to manifest and hidden is less likely to appear.
- "*egg_group_1","egg_group_2*"<- Egg groups are classifications that are used for breeding pokemon.

We have a small but very wide dataset of pokemons, a pokemon is a magical beast with several characteristics. We will also be applying some transformations to the data in order to facilitate the work of our AI algorithms. We should also mention that due to the artificial nature of our dataset there were no missing values (and the ones that were missing codified some information), so most of our changes are to codify the information better. We will explain the changes in the order they appear in preprocessing.r, notice that before those changes we will remove all the variables not listed above.

Another thing to mention is **dummy encoding**[3] : we will be using it for all categorical variables (in some cases we have more than 100 differents categories). This represents the best our variables since the alternative would be assigning numerical values. Assigning numerical values would represent some distances between *types* that would misrepresent information(some *types* being closer is not accurate), thus misinform our models. This happens in *eggtypes* and *abilities* as well.

3.2 Special rows

We will begin by removing some pokemons: *Ditto*[4] and special forms. *Ditto* is a bit of a special case himself, he is a pokemon with very low stats that has the gimmick of copying the pokemon it is against in battle (*stats*, *typing*, etc). He also has his own unique *egg group*, *ability*, and for breeding purposes he can breed with any other pokemon but the resulting egg is of the other pokemon's *species*. Due to all these reasons we consider him an outlier that we must remove. Special forms are either a duplicate or a complete outlier in stats and can't be kept permanently.


Ditto

3.3 Target variable vs categorical variables

We begin by seeing the distribution of the target variable, “*speed*”:

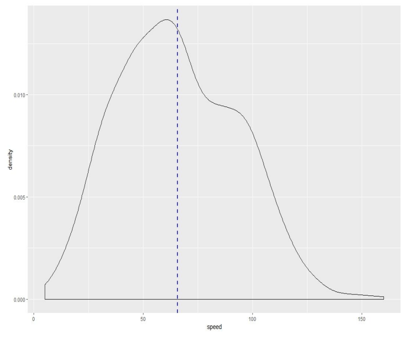


Fig. 1: Density distribution of “*speed*”.

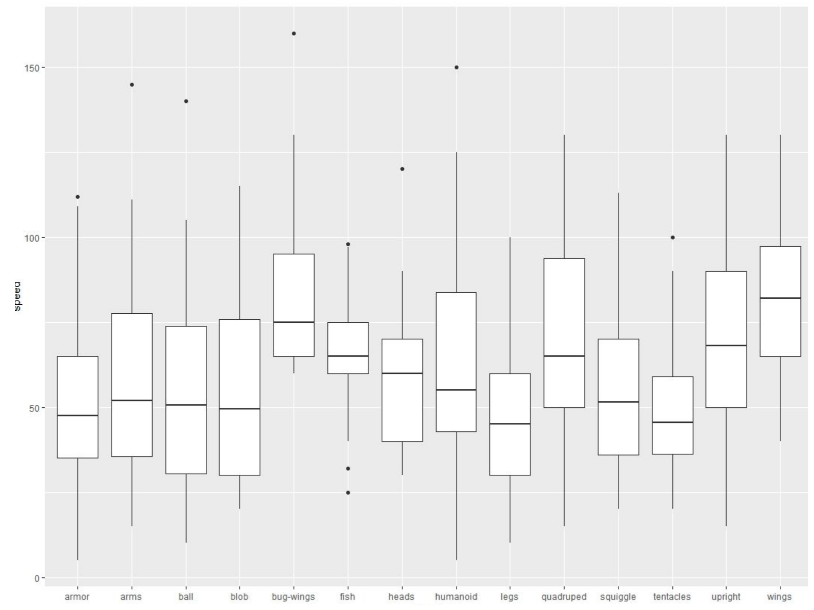


Fig. 2: Boxplot of “*speed*” according to *shapes*.

As we can appreciate in Fig. 1 the distribution seems shaped like 2 gaussians overlapping, so that is a good sign for our AI algorithms.

Grouping the pokemons by their *shape* we can see some heterogeneous pokemon groups. In the Fig. 2 we can observe some facts, pokemons inside the *wings* and *bug-wings* category seems faster than other pokemons. *Quadruped* pokemons also are really fast. In our world, *quadruped* and *flying* animals often are very fast animals, so this can be a good indicator. *Tentacles* are among the slowest, which also makes sense.

We believe that *types* might be very relevant for what we are trying to calculate so we drew a heatmap calculating the "*speed*" mean and standard deviation for *type* combinations:

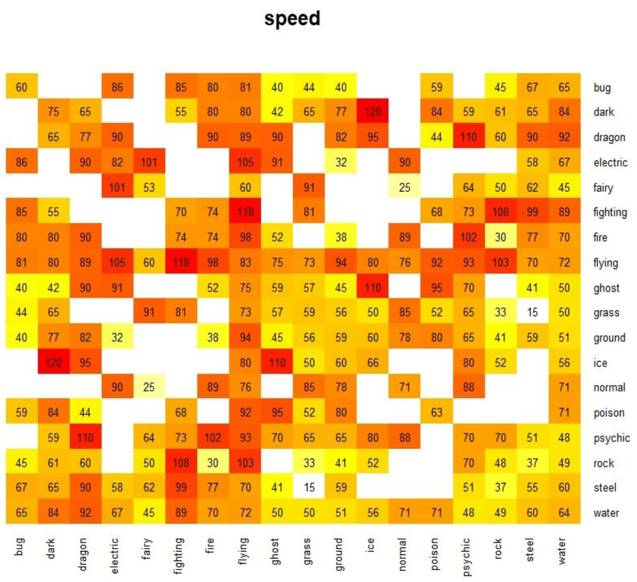


Fig. 3: Mean "*speed*" for pokemon types.

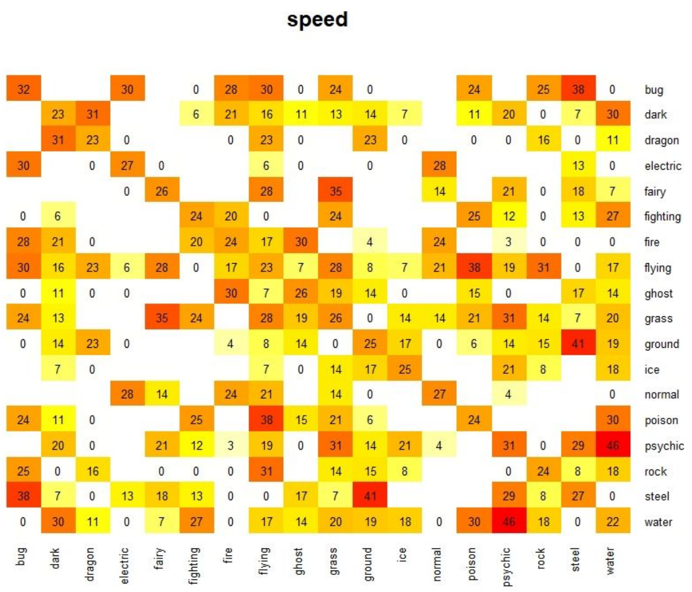


Fig. 4: Standard deviation "*speed*" for pokemon types.

We can see that there are many combinations that only have one instance (sd = 0), and others that have none, which is bad for our algorithms. Among other things, we see that *steel, rock, ghost* and *grass type* tend to have low "*speed*". Meanwhile *fighting, flying, electric* and *dragon* tend to have high "*speed*". We can also see that *bug types* tend to have extreme standard deviation, and *water* and *normal* have a lower standard deviation.

We define a mono *type* as a pokemon that has no dual *type*, here we have plotted a comparison of the mean of the *type* vs the mean of the pokemons that are mono *type*:

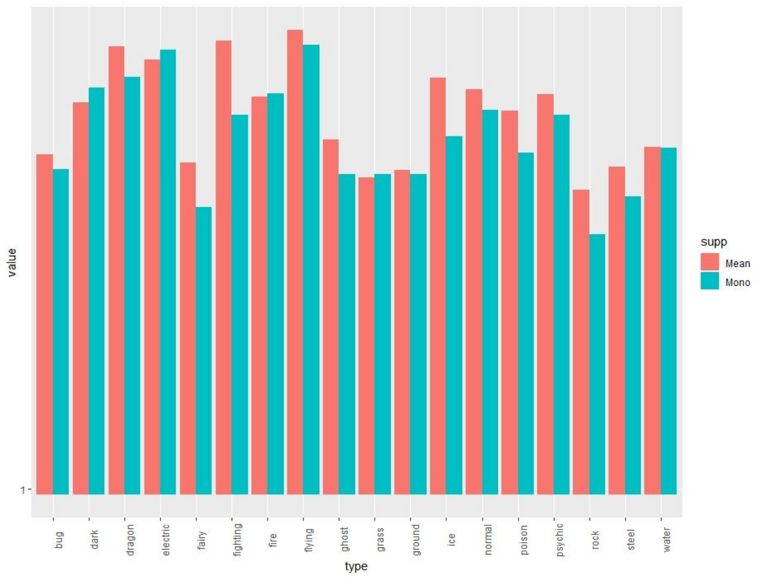


Fig. 5: Mono *type* vs Mean comparison.

We thought this plot would be interesting as we could see how more ‘pure’ *type* pokemon were. We can see that the average *fighting type* is way slower if it doesn’t have a second *type* and the same happens with *fairy* and *rock*. We think that the mono *typing* should better reflect how we expect a pokemon’s “*speed*” to behave due to having that *typing*, which is the information we are after.

Next come the *eggtypes*, which we visualize like *types*. We plot the heatmaps and see if we can find any patterns:

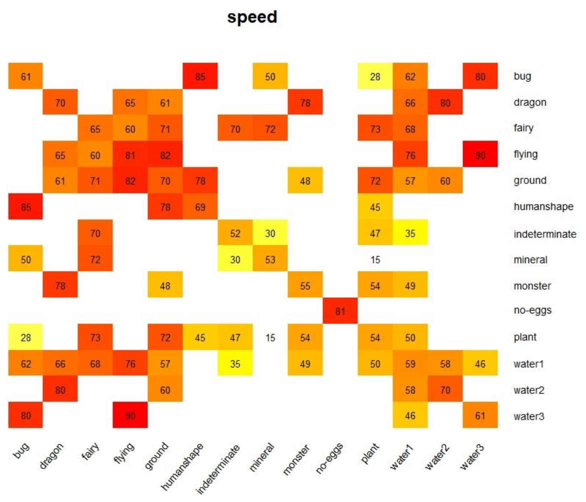


Fig. 6: Mean “*speed*” for pokemon *eggtypes*.

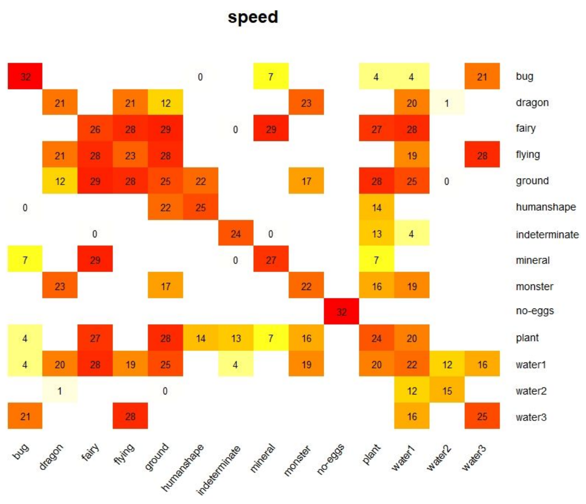


Fig. 7: Standard deviation “*speed*” for pokemon *eggtypes*.

Once again, dragons and flying seem to be a fast *eggtype*, and plants seem to be slow. This time there is also the *eggtype* mineral and monster in the slower ones and fairy and ground seem to be pretty fast as well. Another fact we can see it that mono *eggtypes* seem to have a higher standard deviation, which is surprising and a bit worrying. This means that mixed *eggtypes* better represent a pokemon's "*speed*" than monos.

Finally, we had to deal with the most complicated categorical variables, *abilities*. *Abilities* have the same problems as the earlier variables but with a bit of extra difficulty. We could use **dummy variables** to codify them but if we look at the amount of different *abilities* we have there is a total of 191 different *abilities*.This would mean our dataset would have to add 191 **dummy columns**, which would be ludicrous (more than 200 variables on 720 rows). Not only that, most of those *abilities* are unique so it would be pretty hard to infer data on new pokemon.

After some research we came across a solution: choosing threshold of frequency and bulking all abilities below it. We only keep the information on *abilities* that we are likely to be able to generalize, and significantly reduce the amount of columns generated. We set the threshold to 20 occurrences, and unified the rest of the values with the name "*LOWFREQ*". Then we just applied **dummy encoding**, in the end this implied adding 31 columns.

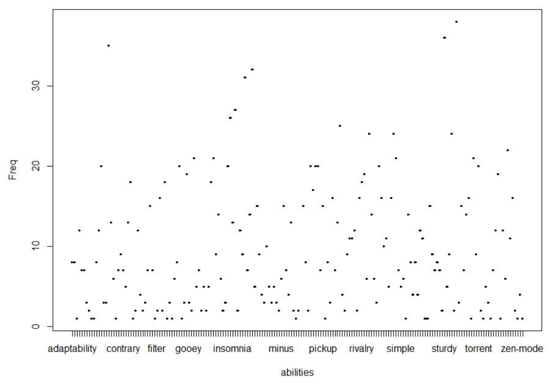


Fig. 8: *Ability* frequency plot.

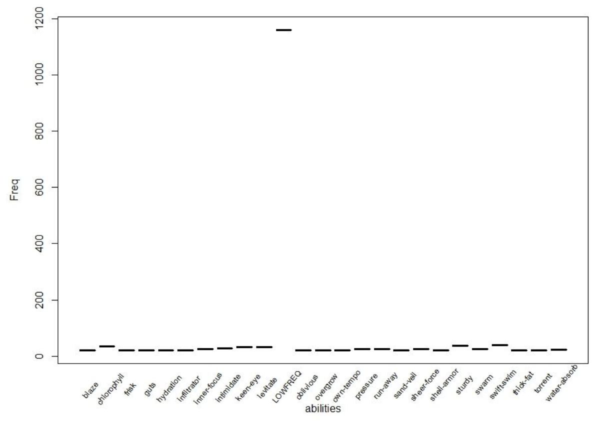


Fig. 9: *Ability* frequency plot after correction.

While the "*LOWFREQ"* category is notably big, we managed to reduce a lot the amount of *abilities*. The rest of the *abilites* have between 20 and 50 occurrences. While this is quite unbalanced it is much better than what we had initially.

3.4 Numerical values correction

Next, we had to correct the stats of the pokemon. After a little research we found lots of different opinions[5,6] about an acceptable value of kurtosis and skewness, but we know that values near to zero are good, so we want to transform our data to get closer to this value. We used 2 methods for transform our data, the square root (only for positive values of skewness) and the logarithm (stronger than square root).

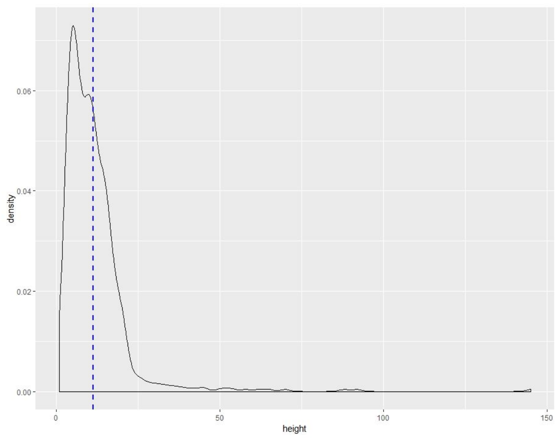


Fig. 10: “*Height*” before correction.

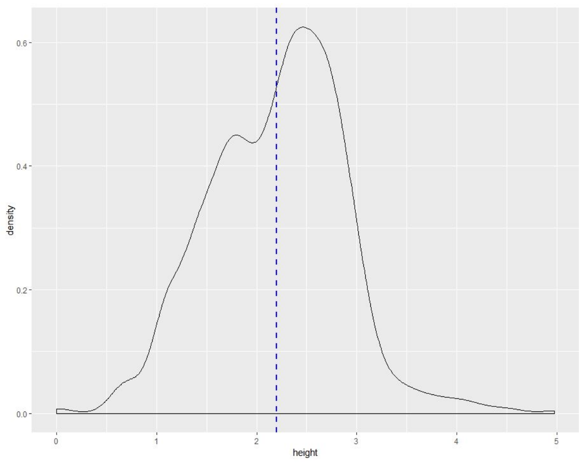


Fig. 11: “*Height*” after correction.

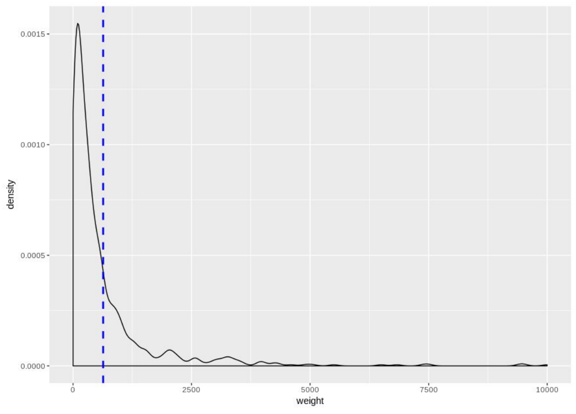


Fig. 12: “*Weight*” before correction.

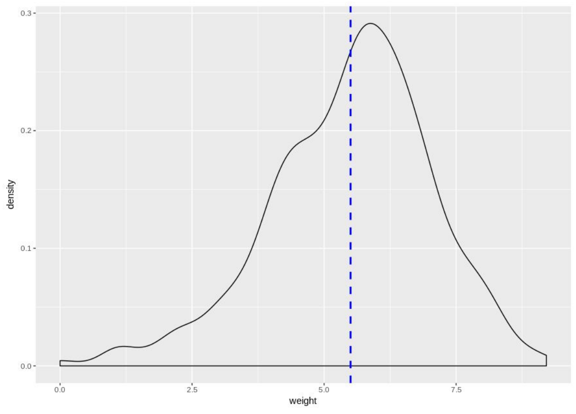


Fig. 13: “*Weight*” after correction.

On the left side we can see that “*height*” and “*weight*” have some very big outliers, that get exponentially farther. This would probably complicate a lot the regression, so we decided that we should apply a logarithmic scale. As we can see on the right side this worked effectively. We apply this procedure (using square root) to all the other numeric variables that needed it and help our methods to achieve a better performance.

At this point, to further facilitate the work of our algorithms we decided to normalize the values of all our numerical values between 0 and 1. This meant diving by the maximum, we noted those maximum values to be able to recover the values our algorithm represents. The normalized columns with their maximum are the following:

- "*height*" 4.977
- "*weight*" 9.159
- "*base_experience*" 6.4102
- "*attack*" 12.8452
- "*defense*" 15.1658
- "*hp*" 15.9687
- "*special_attack*" 12.4097
- "*special_defense*" 15.1658
- "*speed*" 160

4. Data splitting and validation

We have a r code for the creation of our training and test data, in splitting.R we create a random sample of size ⅔ of the whole data for training and the rest for test. We save this 2 datasets for use in the creation of the models later.

We have decided to use a **k-fold cross-validation**[7] with k=4 (except for lm method and mlp). We chose k= 4 because we had 480 pokemons for training: at generation 4 of pokemon there were 493 and that generation has the closest to 480. To make sure our folds are representative, we think that it makes sense to have the average amount of pokemon in a generation chosen as reference.

5. Linear/quadratic methods

We have chosen the **linear regression** and **linear SVM** as linear methods and **quadratic SVM** as a quadratic method.

5.1 Linear regression

The first, Linear regression, is the simplest, we have used this instead of Ridge Regression because of its simplicity (the others two are very complex in comparison due to the large number of variables). This method doesn't require a cross-validation.

5.2 Linear SVM

The second one is **Linear SVM**, as a regression problem we have to use SVM with an extra hyperparameter, **epsilon**. This parameters is the width of the space with no errors. We have decided to keep this parameter constant and try to get good values in the other hyperparameters, in our case only the **cost**.

5.3 Quadratic SVM

The last is **Quadratic SVM**, the polynomial of degree 2. As we have done before we keep **epsilon** constant, the difference is that now we have 2 hyperparameters to adjust, so keeping an extra hyperparameter constant helps saving time to get a result (we can have higher ranges in the other hyperparameters). Another hyperparameter that we keep constant is the **coef0**, which value is **1**. We have done to force a higher feature space.

5.4 Results

Once the **cross-validation** process is finished we obtain the best model with his best set of hyperparameters, and the MSE error from the **cross validation**. The best model use the whole learning set so we want to predict this set with our model. Since we don't use CV with linear regression we only have the **Learning MSE**. We get the following results:

Method	Linear Regression	Linear SVM	Quadratic SVM
CV MSE	0.00958171	0.01568473	0.014001
Learning MSE		0.01056608	0.004245379

As we can see in the table, the best model for now is the **quadratic SVM**, this method has the best **CV MSE** and the best **Learning MSE**.

6. Non-linear methods

6.1 MLP Neural Network

For our first non-linear method we attempted to use a **Neural Network** using the Keras library (that utilizes Tensorflow, for fast calculations). The syntax in R is a bit inusual since this is a library that ports Tensorflow from Python and we must use pipelines to make it work.

After some research and contrasting information[8], we decided to use **relu** as our activation function and leave **linear** for the final value. We fixated the amount of neurons at 40 and tried to work from there. We began to increase the **layers** until we had 5 + output, at which point our results did not improve anymore. We did the same with **epochs** and found out that at **80 epochs** our model barely improved. After documenting a bit ourselves on which hyperparameters to tune first[9] we tested different **learning rates** and **momentum** values. The best values were obtained with a **learning rate of 0.05** and a **momentum of 0.9**. Our **MSE** varied between **0.015** and **0.025** on validation, which is pretty good on normalized results.

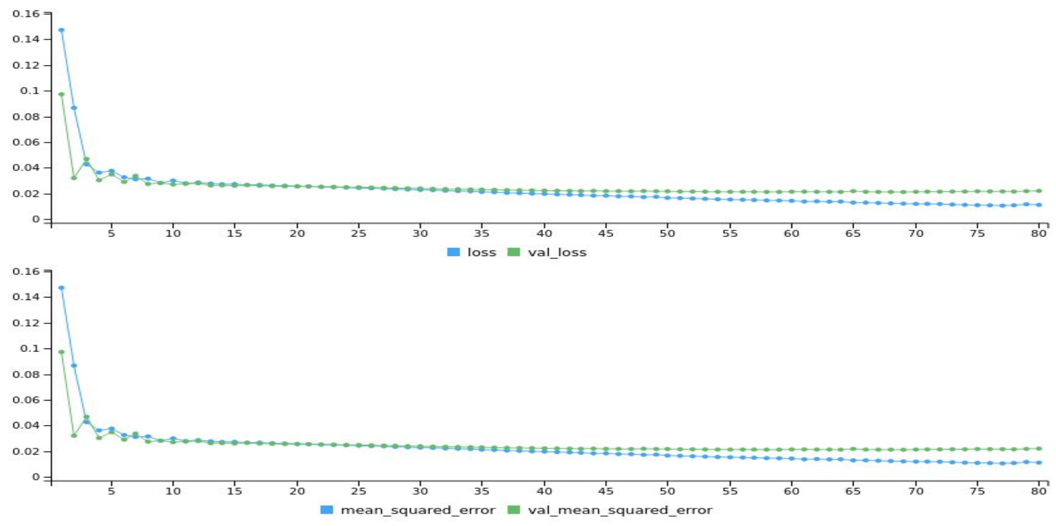


Fig. 14: **Mean squared error** (training is blue and validation is green) during the 80 epochs.

6.2 SVM with RBF kernel

The second non-linear method is another SVM, but now with the **RBF kernel**. The **RBF SVM** have the same hyperparameters than the quadratic so we use the same range in the tuning process.

6.3 Results

Method	MLP	SVM with RBF
MSE Validation	0.015-0.025	0.01392874
MSE Learning		0.003238521

7. Final model

We decide that the best model is the **SVM RBF kernel**, with an **MSE** error of 0.003238521 in the learning, it seems a very good result. The next step is check the **MSE** in our test set to ensure that our model doesn't overfit it.
The **MSE** with the test dataset: 0.0147059.
This result means that the model is overfitted, despite that we considered a good value, in the following fig, we can see the difference between the prediction of our model and the real values. We should comment that there are several weird values.
Looking at the results, most of our test information gets less than 20% error, which gives you an idea of whether a pokemon is fast or not.

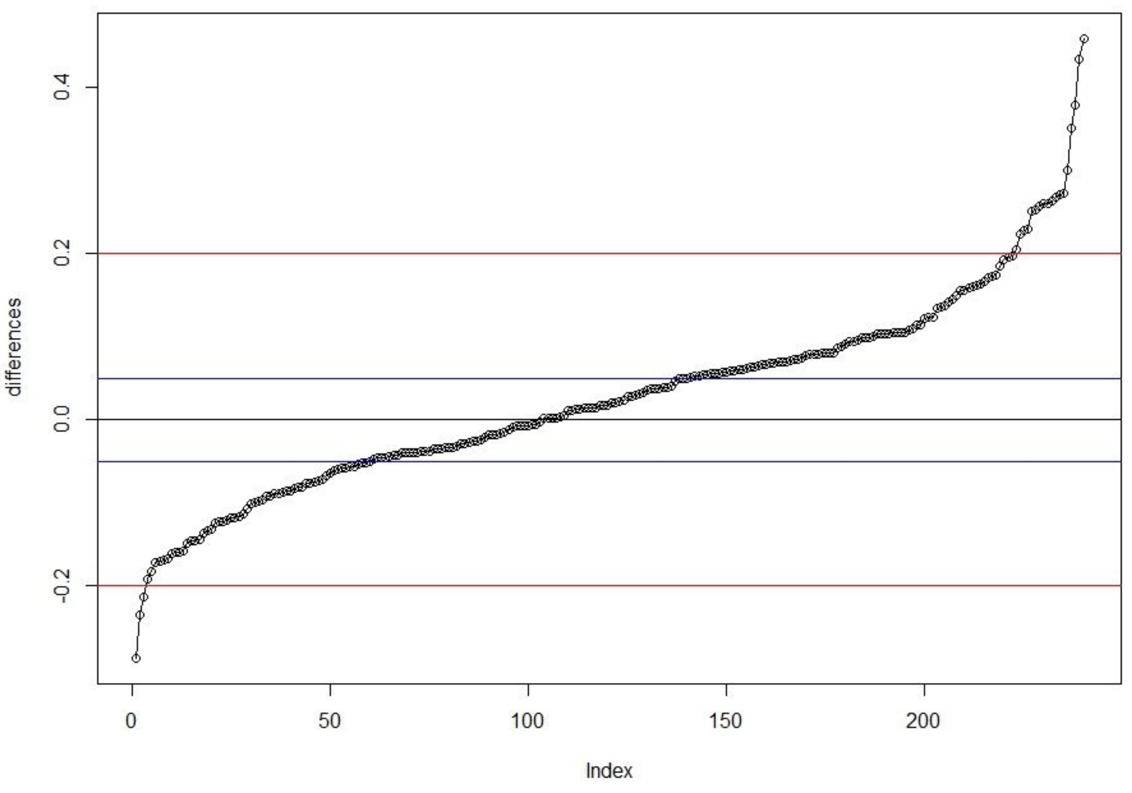


Fig. 15: (predicted-rea) “*speed*” value on the test dataset.

8. A final brief part containing:

8.1. A self-assessment of successes, failures and doubts

We think that we were rather successful for the problem at hand. The main reason is that there are some intrinsic difficulties to the problem we were facing. First of all they are sort of magical creatures that would not necessarily relate to their physical attributes (there are many incoherences when looking at a pokemon and seeing its stats). Second of all and more important is the fact that when a new pokemon is designed it is meant to be new, both physically and stats-wise. This means that we are trying to generalize a problem which tries not to be generalizable. For those reasons, we consider that an error of 0.0147059 is a big success.

A possible failure has been including the *types* and *eggtypes* that had none or one pokemon. Since diversity is rather low in those fields we should have probably set a threshold for pokemon types and put only the combinations that had some frequency (like we did with abilities). To reinforce this, we have tested on mlp using only our numerical variables (none of the dummy variables) and have obtained better results, which may be due to the noise they add.
Another failure could have happened when we decided to keep **epsilon** constant. We don't explore other possibilities and we could have found another better combination of hyperparameters for our data.

#doubts
We found that values between +2 and -2 of **skewness** and **kurtosis** are good enough, but also we found that values above +1 or below -1 are considered bad in **skewness**, this is not a real problem because of with the transformation we can achieve very good values. Another doubt is about use more than one transformation in the same dataset, we think that is not a problem but we are not sure due to some post that considers that it's an error in the process of treatment of the data.

8.2. Scientific and personal conclusions

Dummy variables are very powerful, specially for when we guarantee completely independent factor levels. Preprocessing is at least 50% of the work in machine learning, and should be dedicated a considerable amount of time. Some of the preprocessing improvements we have made have been posterior to implementing the models and they have significantly improved our results.

Pokemons do have some patterns to be figured. When we started this project, we were uncertain whether we would be able to have any success in it. Surprisingly we obtained some reliable conclusions. Since the physics are 'sort of similar' to the real world we expected to have some results but we did not expect getting such a low error.

There is no easy way to deal with new pokemon outliers. The data we had at the beginning was quite limited on them and we clearly reached a hard capping for our error. If we had to guess, this is probably the pokemons that were introduced in the last generation and were unique in their gimmicks. This made our error very high when predicting them.

Another small conclusion is that many variables are gaussian distribution after changing their space a bit, and this can help predicting them.

8.3. Possible extensions and known limitations

A fun extension of this model would be to try it again on the new generation, since we used pokemon up to gen 6 and gen 7 is already out. If our conclusions are correct, we should observe an error slightly higher than the one obtained in testing, there should be new pokemon to cover new roles that the model hasn't trained onto.

A strong limitation of this model is, as mentioned in failures, predicting using very weak variables. Unique pokemon typings are going to be predicted with a lot of noise for new pokemon. There is also a risk that some abilities don't appear in the training and thus the model has no information on them. Even more so in the models that don't use 4-fold cross validation.

The visualization done on this project could also help with understanding what roles/ type combinations are lacking in the pokemon universe, and what to expect from each categorical value of a pokemon (you could usually get its type according to their looks). For starting players that like to play the games optimally this information is probably interesting.

The next step in our project we would attempt to correct the mistakes done due to including categories with very little representation (using the threshold). We could also set a threshold to filter variable combinations with high standard deviation. Some bootstrapping may also help our algorithm generalize better, as 700 instances are very few.

For curiosity's sake we use the test set in the other models, we discovered that the MSE is very similar to our best model MSE(0.0147059):

Method	Linear Regression	Linear SVM	Quadratic SVM
TEST MSE	0.01595128	0.01629096	0.01530435

We can conclude that there aren't a better performance due to the outliers pokemons. For gaming purpose this pokemons are different to another similars to get variability, this pokemons usually have extreme stats and don't have any pattern.

9. Bibliography

[1] Parr, D. (2019). *Pokedex*. [online] Kaggle.com. Available at: https://www.kaggle.com/davidrgp/pokedex [Accessed 11 Jan. 2019].
[2] González, A. (2019). *Pokemon EDA | Kaggle*. [online] Kaggle.com. Available at: https://www.kaggle.com/donyoe/pokemon-eda [Accessed 11 Jan. 2019].
[3] En.wikiversity.org. (2019). *Dummy variable (statistics) - Wikiversity*. [online] Available at: https://en.wikiversity.org/wiki/Dummy_variable_(statistics) [Accessed 11 Jan. 2019].
[4] Bulbapedia.bulbagarden.net. (2019). *Ditto (Pokémon) - Bulbapedia, the community-driven Pokémon encyclopedia*. [online] Available at: https://bulbapedia.bulbagarden.net/wiki/Ditto_(Pok%C3%A9mon) [Accessed 11 Jan. 2019].
[5] Medium. (2019). *Day 8: Data transformation — Skewness, normalization and much more*. [online] Available at: https://medium.com/@TheDataGyan/day-8-data-transformation-skewness-normalization-and-much-more-4c144d370e55 [Accessed 11 Jan. 2019].
[6] Anon, (2019). [online] Available at: https://www.researchgate.net/post/What_is_the_acceptable_range_of_skewness_and_kurtosis_for_normal_distribution_of_data [Accessed 11 Jan. 2019].
[7] Brownlee, J. (2019). *How To Estimate Model Accuracy in R Using The Caret Package*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/ [Accessed 11 Jan. 2019].
[8] Singh Walia, A. (2019). Activation functions and it’s types-Which is better?. Retrieved from https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f
[9] Yak-Tak Ng, A. (2019). Tuning Process (C2W3L01). Retrieved from https://www.youtube.com/watch?v=AXDByU3D1hA&index=24&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc