POKEMON DATASET

Evaluating and Improving the Model

1. Legendary Pokémon Classifier

Bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification. It works like this:

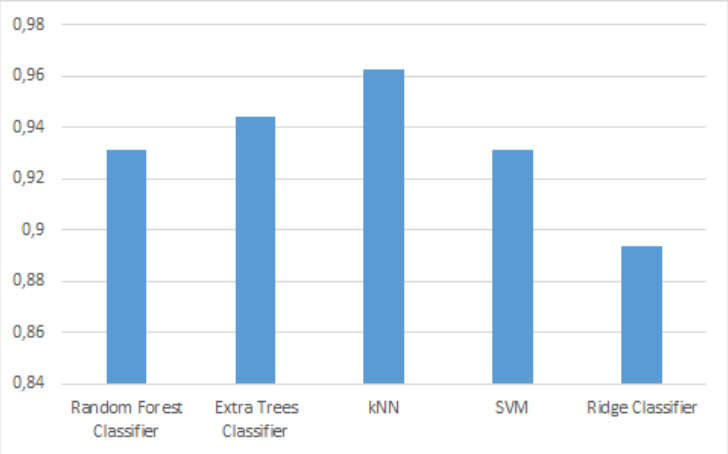
1. We pick n sets from our learning set, as in bootstrap.
2. For each of the sets we build a partial classifier.
3. We add the voting mechanism.

We decided to evaluate our Legendary Pokémon Classifier by building Bagging Classifier on several classification algorithms, and decide which one is the best. Results are shown in the table below.

Algorithm	Score
Random Forest Classifier	0.93125
Extra Trees Classifier	0.94375
kNN	0.9625
SVM	0.93125
Ridge Classifier	0.89375

Created by: Grzegorz Meller, Mateusz Rock, Tomasz Wierciński, Robert Zwierzycki

POKEMON DATASET



Previously (in the last report), we used only kNN classifier which gave result 0.954. We can clearly see, that choosing kNN was a good choice, other classification algorithms would not give such a high result. Bagging also improved a little our kNN algorithm, by setting n=5 (previously we put n=3). As we see on the table it gives a bit better results, but we need also to remember that in bagging training data are chosen randomly, so it also can be the case.

Bagging came out to be very fast, convenient and optimal method of building classifiers, because without special knowledge about them, like setting their parameters, bagging classifier will automatically set all hyper-parameters for us in the most optimal way.

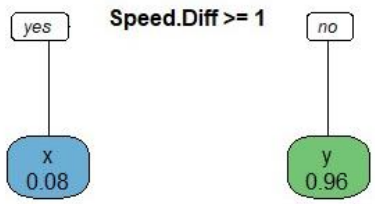
Python code:

```
seed = 1075
np.random.seed(seed)

rf = RandomForestClassifier()
et = ExtraTreesClassifier()
knn = KNeighborsClassifier()
svc = SVC()
rg = RidgeClassifier()

bagging_clf = BaggingClassifier(rf, max_samples=0.7, max_features=30, random_state=seed)
bagging_clf.fit(X, y)
print(bagging_clf.score(X_test, y_test))
print(bagging_clf.get_params())


bagging_clf = BaggingClassifier(et, max_samples=0.7, max_features=30, random_state=seed)
bagging_clf.fit(X, y)
print(bagging_clf.score(X_test, y_test))
print(bagging_clf.get_params())

bagging_clf = BaggingClassifier(knn, max_samples=0.7, max_features=30, random_state=seed)
bagging_clf.fit(X, y)
print(bagging_clf.score(X_test, y_test))
print(bagging_clf.get_params())

bagging_clf = BaggingClassifier(svc, max_samples=0.7, max_features=30, random_state=seed)
bagging_clf.fit(X, y)
print(bagging_clf.score(X_test, y_test))
print(bagging_clf.get_params())

bagging_clf = BaggingClassifier(rg, max_samples=0.7, max_features=30, random_state=seed)
bagging_clf.fit(X, y)
print(bagging_clf.score(X_test, y_test))
print(bagging_clf.get_params())
```

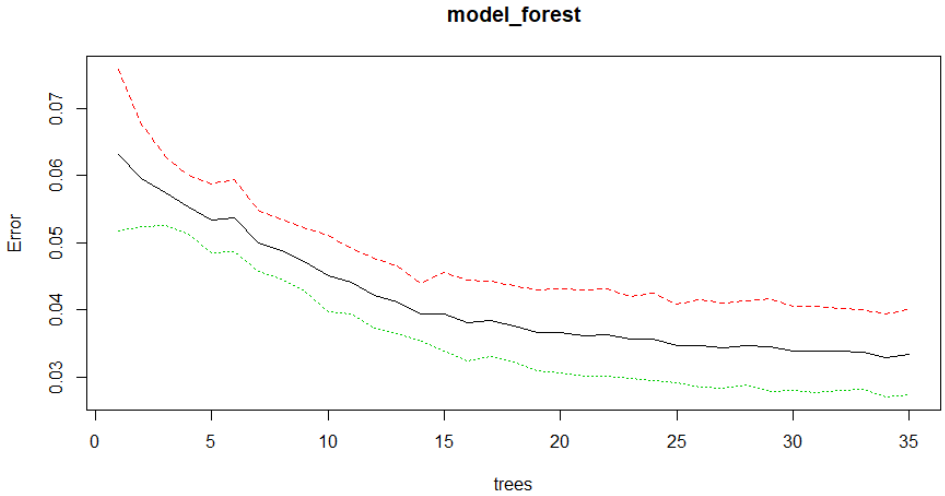
Created by: Grzegorz Meller, Mateusz Rock, Tomasz Wierciński, Robert Zwierzycki

POKEMON DATASET

2. Predicting results of future combats



The initial model used for the classification task was a simple decision tree, which was built on a single attribute - speed difference between opponents. Despite its simplicity it performed remarkably well and resulted in 94% accuracy predictions on the testing set. The accuracy testing was performed on multiple random divisions of the whole set into training and testing sets. The next step was improving on the model. We used a modified bagging ensemble called random forest. We observed as the estimated out-of-bag error rate changed along with the number of classification trees used in the forest.



The red line shows the x class error and the green one - y class error. The black line is the out-of-bag error rate. Combats classified with “x” are the ones in which the attacking Pokémon was named the victor. The “y” class is used when the defending Pokémon wins. The almost constant difference in error between the two classes could mean that the model still doesn’t take into account the order of attacks.

Finally we have decided to use 35 classification trees in our forest. That resulted in an out-of-bag error rate equal to 3.33%, and an accuracy of 96.8% when predicting on the testing set. It’s important to note here that the random forest approach is much more susceptible to changes due to randomness. The split of the set and the inherent randomness in the way the random forest algorithm select features. The accuracy value was therefore much more susceptible to change. A mean from multiple tests was taken to counteract that.

The resulting classification trees still mostly take into account the speed difference. The table below shows the importance of every attribute.

	x	y	MeanDecreaseAccuracy	MeanDecreaseGini
Type.1.x	0.023626958	0.014033314	0.018569358	365.64125
Type.2.x	0.026667296	0.011895843	0.018877847	334.63821
Type.1.y	0.026675387	0.013491734	0.019720497	325.77268
Type.2.y	0.021906018	0.015138982	0.018339545	336.21999
Mult	0.004826597	0.002606091	0.003652174	97.66618
HP.Diff	0.006744376	0.004776175	0.005708075	198.28820
Attack.Diff	0.054450332	0.032955979	0.043115942	566.99480
Defense.Diff	0.004783639	0.004517441	0.004645963	166.34060
Speed.Diff	0.422513527	0.359465412	0.389252588	8989.85512
Sp.Atk.Diff	0.005233964	0.002665144	0.003877847	268.04067
Sp.Def.Diff	0.003146230	0.002426449	0.002768116	163.20915

POKEMON DATASET

Confusion matrix for prediction on testing set

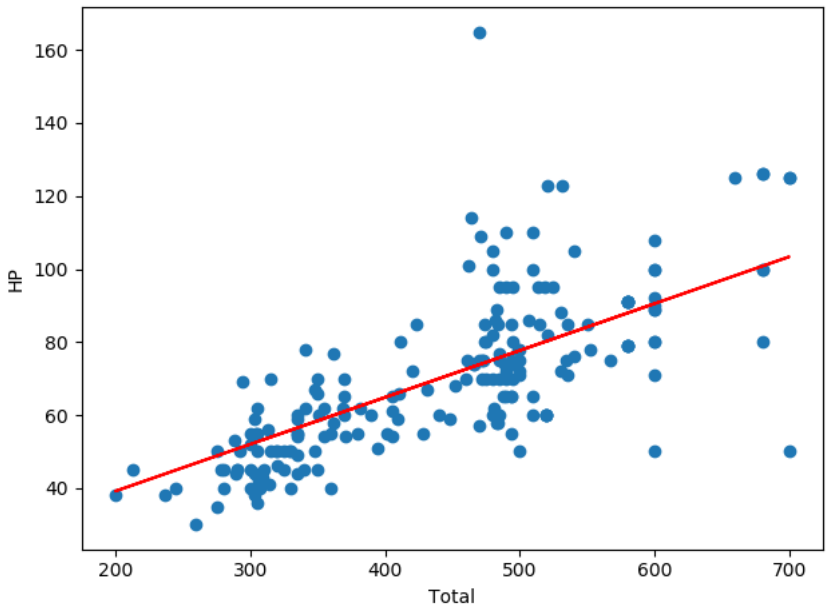
	Reference	
Prediction	x	y
x	5663	165
y	229	6443

3. PREDICTION OF THE HP OF POKEMON WITH LINEAR REGRESSION

Our team has conducted an experiment with predicting the Hp of the Pokémon's, according to other parameters, like (attack, speed, defence, etc.) based on a linear regression model.

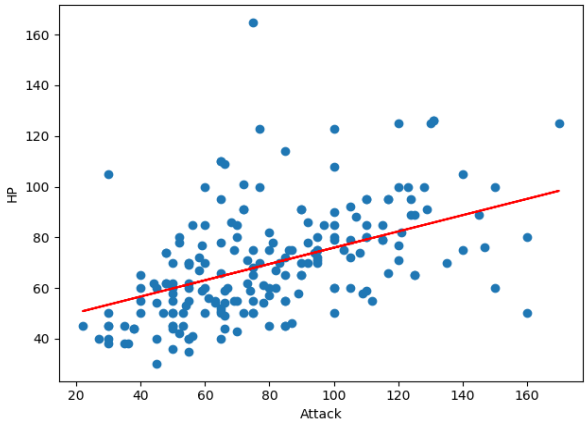
The term "linearity" in algebra refers to a linear relationship between two or more variables. If we draw this relationship in a two-dimensional space (between two variables), we get a straight line. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y (output). If we plot the independent variable (x) on the x-axis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points, as shown in the figure below.

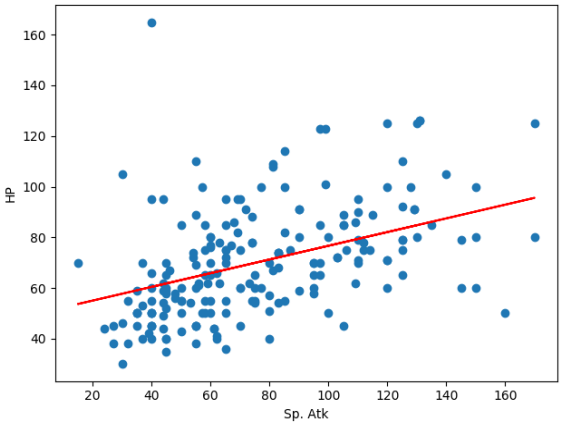
Below are six plots each shows the regression line from an another independent value (attack, defence, speed, etc.) and our target value - Hp of the Pokémon. The blue dots show all the test samples from the dataset and the red line it is the line/function, which best fits the data.

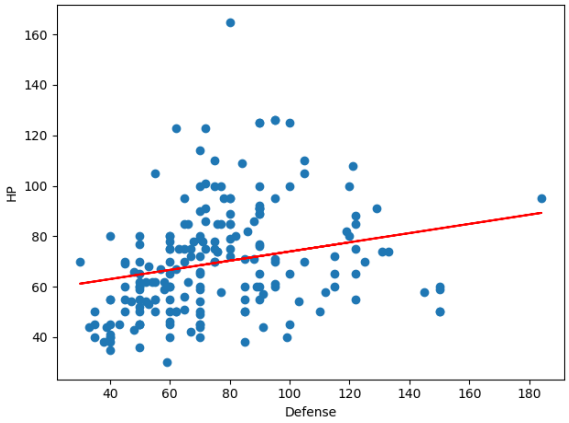


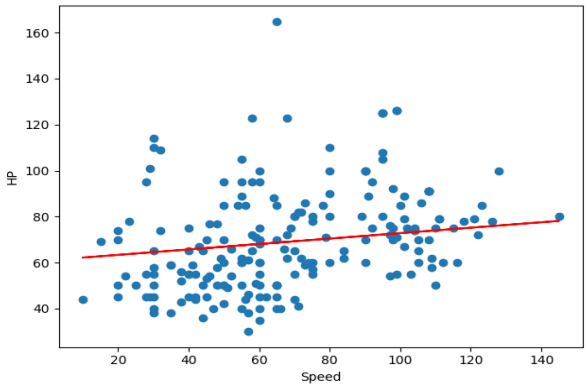
Created by: Grzegorz Meller, Mateusz Rock, Tomasz Wierciński, Robert Zwierzycki

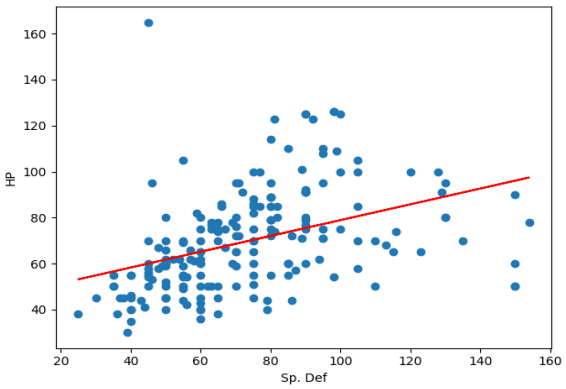
POKEMON DATASET












```
model = linear_model.LinearRegression(normalize=False)
model.fit(train_X,train_Y)
```

```
prediction = numpy.reshape(prediction, 200)
prediction = floor(prediction)
print(accuracy_score(test_Y,prediction))
```

We trained our linear regression model on 600 (75% of dataset) samples, and tested it on 200 (25% of dataset) samples .The accuracy from predicting the Pokémon's Hp is 72, 5%.

Created by: Grzegorz Meller, Mateusz Rock, Tomasz Wierciński, Robert Zwierzycki

POKEMON DATASET

4. Pokémon dream team

We have chosen the best six Pokémon to the dream team based on number of won combats. The improvement in this case is enlarging the data set on which we made it. We used set of 50 000 combats and 10 000 of predicted by us combats. Our results are similar to previously obtained, however there appeared 3 new Pokémon's. Now, the dream team consist of:

5. The strongest and the weakest Pokémon

We have chosen the strongest and the weakest Pokémon based on number of won combats. To find the strongest and weakest Pokémon overall we have chosen a subset of 50 000 combats that have occurred and 10 000 of predicted by us combats. The weakest Pokémon has changed to Shuckle. The strongest one is still Mewtwo.

Created by: Grzegorz Meller, Mateusz Rock, Tomasz Wierciński, Robert Zwierzycki