Group Members: Jonathan Wu, Jason Huang, Taiyo Nakai

DS 110 Final Project Essay

Introduction

The prominent issue of determining the name of Pokemon is a widespread issue for our generation. Individuals who lack the ability to either visually or characteristically discern Pokemon are at risk of ridicule and embarrassment. When asked, teens often recall memories of struggle and failure when they played the game “Who's That Pokémon?” online. Failure is statistically less prevalent in the easy mode of the game where the complete picture of the Pokemon is provided. In an attempt to solve this problem, we have created a machine-learning model that predicts Pokemons’ names based on images. The machine-learning model could improve a player’s accuracy in the game by informing the player of the correct answer. This algorithm is useful if we were tasked to report the names of Pokemon based on given images.

Based on our proposal feedback, we established to make our model discriminate between 5 kinds of Pokemon: Bulbasaur, Squirtle, Charmander, Pikachu, and Mew. We noticed the random forests classifier and k-nearest neighbors classifier is less statistically significant as we increased the size of the categories. By scaling back our proposal of classifying all generation one pokemon, which has a category size of 151, we would increase the accuracy of our classification.

Methodology

The objective of our project is to train the machine-learning algorithm to be able to identify any first evolution starter Pokemon including Pikachu and Mew from a given image. We applied two approaches to predict the Pokemon with machine learning: random forests and k-nearest neighbors. Random Forests will be one approach, as their efficiency in using multiple decision trees is attractive for training an image classifier. K-Nearest Neighbors will be another approach, given its simplicity. While it may not be the best for our purpose, a classifier using KNN could be compared with that using Random Forests to hopefully confirm and teach us that the latter is better. We used an image database from Kaggle that contains images of each Generation 1 Pokemon to train our machine-learning algorithm. Each Pokemon has a folder that contains an image count from 41 to 298, with 10,000+ in total. We trained our model based on 5 folders of Pokemon (Bulbasaur, Squirtle, Charmander, Pikachu, and Mew).

Our code requires importing the library “os” and module “Path” from “pathlib” to access our directory of images. The “Image” module from “PIL” is used to access and extract information from our images. Then, the “numpy” module is used to store our images’ data as arrays. Finally, we use the Bunch model from scikit-learn to create Bunch objects as data for Machine Learning. To split our data into training and testing data, we use the “train_test_split” module from scikit-learn. To create our machine learning model, we import the modules “KNeighborsClassifier” and “RandomForestClassifier” from scikit-learn.

To begin, we process and normalize all the images in our dataset. This requires the code to go through every file (image) in our directory. We had to remove some files for efficiency because they are neither a png or jpg file. We normalize our images by resizing them to a 64x64 size. We then convert all images to RGB to remove alpha channels which indicate whether an

image pixel is either fully transparent or opaque. To signify that the new image file is normalized, we append the string “new” to the name of the file. For each image, we then convert each image file into a PNG to ensure that there are no discrepancies between each image file.

Secondly, we create machine learning data from our normalized images. This requires the code to go through every file (normalized image) in our directory. We use the module “Image” from “PIL” to extract rgb values from the images as lists. We translate this data into a Bunch object because that is indicated in our lecture and documentations about the machine learning algorithms.

Next, we train our machine learning model based on two classifiers: random forests and k-nearest neighbors. To split our data into training and testing data by using the module “train_test_split”. We use a standard random_state = i so we can measure the difference in the parameters for each case. We deduce a test size of 0.15 seems like a fair amount given the amount of images that we are working with. The parameters we are manipulating for random forests are n_estimators, criterion, and min_samples_leaf. The parameters we are manipulating for k-nearest neighbors are: n_neighbors, leaf_size, and weights. The changes in parameters allow us to determine which factors have the most significant influence on accuracy.

To keep record of the efficiency of our code, we use the “time.perf_counter()” function to keep track of time for each step. Also, to determine the precision of our algorithm, we altered the number of trials.

Results

Processing and Normalizing Images

- Time elapsed to process and normalize data: 24.302 seconds!

Creating Data From Images

- Time elapsed to create data: 11.4738 seconds!

Training RandomForest Classifier #1

- Testing RandomForest with 85.91% accuracy in 3.7848 seconds with 100 trees!
- Testing RandomForest with 87.27% accuracy in 11.2050 seconds with 200 trees and entropy!

Training kNearestNeighbor Classifier #1

- Testing kNN with 69.55% accuracy in 0.0005 seconds with 7 neighbors, leaf size 100, and distance method!
- Testing kNN with 64.55% accuracy in 0.0002 seconds with 5 neighbors!

Conclusion

- Over 1 trial, default RandomForest has an average of 85.91% accuracy.
- Over 1 trial, modded RandomForest has an average of 87.27% accuracy.
- Over 1 trial, default kNN has an average of 64.55% accuracy.
- Over 1 trial, modded kNN has an average of 69.55% accuracy.

- Over 10 trial, default RandomForest has an average of 84.32% accuracy.
- Over 10 trial, modded RandomForest has an average of 85.23% accuracy.

- Over 10 trial, default kNN has an average of 63.55% accuracy.
- Over 10 trial, modded kNN has an average of 72.36% accuracy.

Conclusion

From even one trial run, we can see that random forest is generally the better algorithm for handling images than k-nearest neighbors by over 15 points every time. On average, random forests has an accuracy of about 85% for our model given the default parameters.

Link to Github: https://github.com/capitalhere/DS110_Pokemon