ELEC 301
Final Project - Pokemon Type Prediction

Report Submitted: December 13, 2017

Daniel Vadasz, Robin Kim, Carl Henderson

Contents

1 Introduction

From spam filtering to medical diagnosis, machine learning is starting to become leveraged in every part of our lives. Today's machine learning is not the advent of sentient artificial intelligence but rather a tool to help us make sense of the ever expanding data stores of the modern world. Machine learning is currently a field of study interested in the development of algorithms to make intelligent actions based on given data [1]. As learned in class, these algorithms work by taking labeled or unlabeled training data x_i and attempting to infer a function $f(x_i)$ such that

$$y_i = f(x_i)$$

The two types of machine learning problems that exist are supervised learning and unsupervised learning. Supervised learning is where labeled data is given and a predicted y_i is wanted, while unsupervised learning when unlabeled data is given and information from the data is discovered. Some popular machine learning algorithms include K-Nearest Neighbors, Support Vector Machines, and Random Forest which we will cover later on in the report.

Since these these algorithms require a trove of data in order to make any intelligent actions, we need to first identify problems or tasks that can be learned through data. Luckily, today's world is filled with plenty of data. One source of data is from the popular video game called Pokemon where in-game characters are classified as types. Therefore, the primary objective of this report is to outline our use of machine learning algorithms to classify Pokemon types from given data and images.

2 Problem Description

The problem we were tasked with for this project was to develop a machine learning model to classify the type of a Pokemon given an image of the Pokemon as well as some metadata such as stats of the Pokemon. Therefore, this is a supervised classification problem. We are given 601 randomly chosen Pokemon for our training data and have 201 Pokemon to be judged on for our ability to classify type correctly. Our predictions for the 201 Pokemon will be judged through an online Kaggle competition.

Since there exist 18 types of Pokemon, we have to tackle an 18 class classification problem which can be quite difficult, especially if we lack sufficient data or the data isn't a suitable predictor for the class. Therefore, a decent accuracy score should be in the 30% range, which we will try to achieve with our machine learning algorithms. We will likely need to use both the image and metadata stats in order to achieve better classification results.

3 Data Exploration and Visualization

After being introduced to the problem and its objective, the next step we took was to examine the given data in detail, which is known as the exploratory data analysis process in data science. In this section, we explore the data's features and examples and try to notice any trends or peculiarities in the data about the Pokemon.

We began the exploratory data analysis process by loading in the Pokemon metadata csv files. According to the data descriptions of the given csv files, we have 601 rows of training data and 201 rows of test data which both have 15 predictors to describe each Pokemon. In addition,the training data includes an extra column giving the Pokemon type. The 15 predictors include Pokemon stats such as HP, Attack, Defense, Speed, Total, and many more. An example of the first five training Pokemon rows are shown in Figure 1 on the next page.

	Type	HP	Attack	Defense	SP Atk	SP Def	Speed	Total	HPEV	AtkEV	DefEV	SpAEV	SpDEV	SpeEV	Mass	Height
0	5	45	49	49	65	65	45	318	0	0	0	1	0	0	6.9	71
1	5	60	62	63	80	80	60	405	0	0	0	1	1	0	13.0	99
2	5	80	82	83	100	100	80	525	0	0	0	2	1	0	100.0	201
3	2	58	64	58	80	65	80	405	0	0	0	1	0	1	19.0	109
4	3	44	48	65	60	54	43	314	0	0	1	0	0	0	9.0	51

Figure 1: Screenshot of Pandas Table of first five Pokemon and their stats

Following this, we decided to look at the type data more in depth. We observed that there are 18 unique Pokemon types. Therefore, we will have to predict a class 1-18 for each our test Pokemon. After plotting a histogram of the frequency of each class, we saw that there is not a uniform distribution of each class with class 3 being the most common with 82 Pokemon. Our histogram of the frequency vs. Pokemon type is shown in Figure 2 below. That lead us to believe our predictions should likely not be uniformly distributed and contain all types.

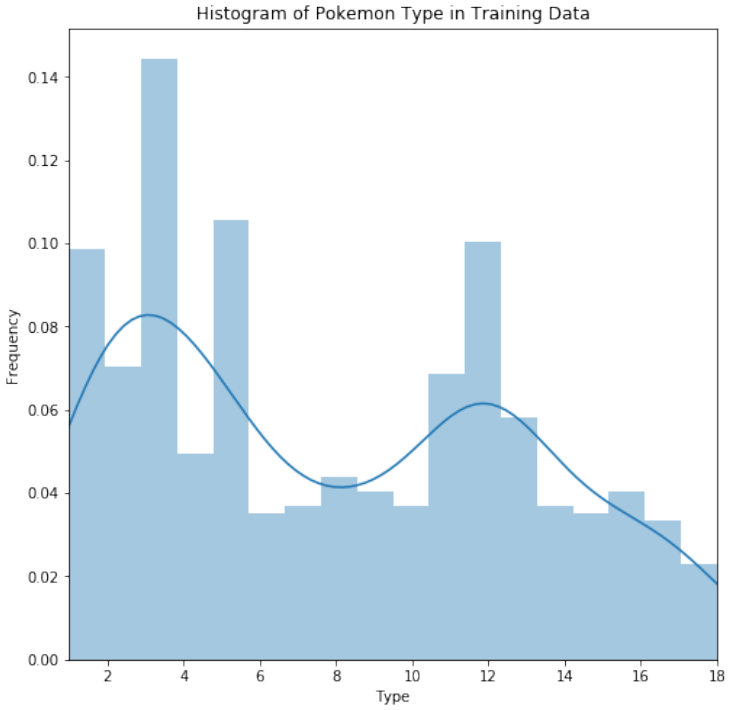

Figure 2: Histogram of Pokemon type from training data

After exploring the type data, we decided to look into the 15 predictors given for each Pokemon. We noticed that every predictor is given as a integer rather than other data types such as a string. We decided to create a correlation matrix of the training data to see if there is any strong correlation between any two stats. From the correlation matrix, we can see that many of predictors are not correlated such as Speed and Defense while some predictors are correlated such as Mass and Height. From these results, it is likely that we can perform some dimension reduction methods to our data such as Principal Component Analysis. The correlation matrix is shown in the Figure 3 on the next page.

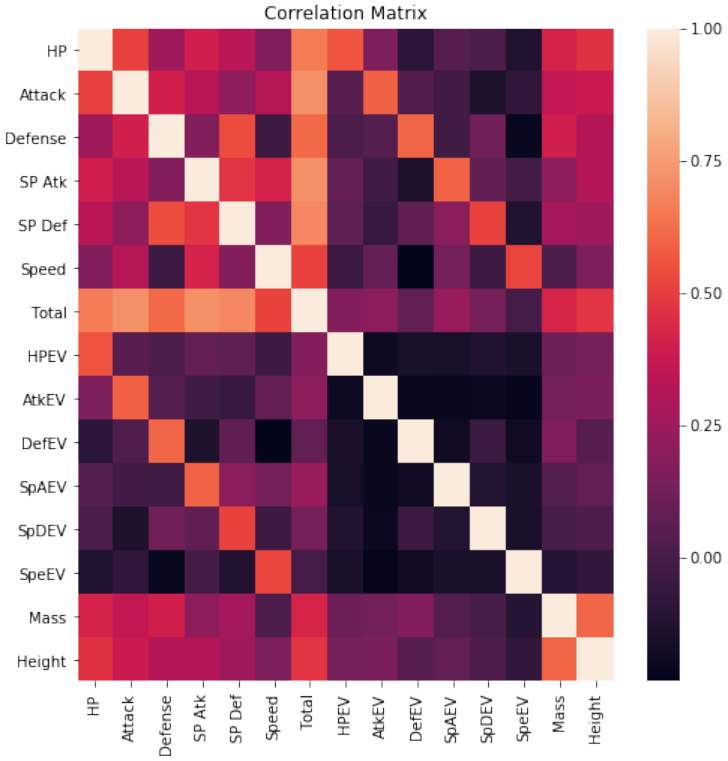

Figure 3: Correlation Matrix of training data

Although stats may help better describe the Pokemon's attributes during combat, pictures of Pokemon may help us classify the Pokemon type with greater accuracy. Therefore, we decided to observe some of the first first given training images as well as the test images. Each of the images are 96x96x3 due to them living in the RGB colorspace, meaning we have up to 27,648 predictors for each image, which is much more than our 601 training samples. Thus, we will likely need to extract information such as the most prevalent colors in the Pokemon as well as the relative size of the Pokemon. The images will likely require more pre-processing work than the csv data. An example of one of the training Pokemon images is shown in Figure 4 below.

Figure 4: First training Pokemon image

The exploration of the Pokemon metadata as well as the images of the training and test Pokemon allowed us to get comfortable with the data we were given as well as identify possible next steps in pre-processing and feature extraction.

4 Preprocessing Steps Taken

We started prepossessing by creating a framework for our Python program that included basic elements such as imported libraries, setup of X_training data matrix, y vector, X_test_data matrix, scaling, cross-validation, and submission file creation. We imported libraries commonly used for machine learning such as numpy and pandas. The X_training data matrix was created by dropping the 'Type' and 'Number' columns from the given TrainingMetaData dataset. We dropped the 'Type' column because we wanted to include the 'Type' column in the y vector for training purposes. The 'Number' column was not included since we assumed it did not provide useful information to our model to predict the type of the Pokemon in question. The y vector is the 2nd column of the given dataset which corresponds to the type of Pokemon. Using the sklearn library, we imported standardscaler in order to scale the X_training and X_test matrices. Cross validation was included in the framework to give us more accurate results. Finally,the last thing in the framework is the submission file creation which turns the predicted y values of the X_test data into a .csv file.

5 Dimensionality Reduction

We also experimented with both LDA and PCA to reduce the dimensionality of the data matrices since we believed that some of the stats did not contribute much to the prediction of type and would only add noise to our model. A three dimensional representation of the data achieved via PCA is shown in Figure 5 below. Since there was no obvious grouping of the data in three dimensions, we assumed the data lived in a higher dimensional space. We decided to use LDA, Linear Discriminant Analysis, because as a supervised learning algorithm it computes directions that will maximize separation between the 18 classes or types in our case. Though it serves as a classifier, we chose LDA for its dimensionality reduction property. For example, when used with Logistic Regression, LDA + Logistic Regression outperformed PCA + Logistic Regression by 8%. LDA works by projecting the N-dimensional space where N is 601 for our training data and projecting the data onto a C-1 dimensional space where C is the number of classes or types in our case. LDA attempts to maximize the ratio of between-class scatter to within-class scatter.

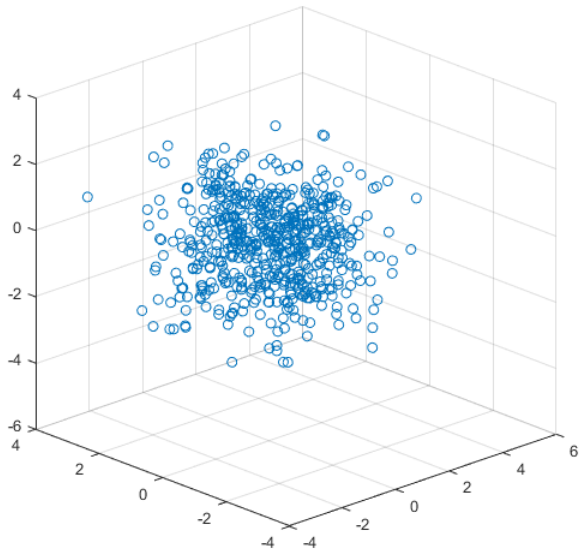

Figure 5: PCA projection onto a three dimensional space

6 General Feature Selection

Instead of using the entire image vectors as features for our algorithm, which would drastically increase the size of our data matrix and likely slow our computations, we decided to extract only the most useful features from the images. The first feature we attempted extracting was color, which is likely the most significant visual indicator of a Pokemon's type. This feature extraction was performed via the Python library "colorgram." For each of the training/test images, we extracted the four most common colors (not including black) via their RGB values and added them as 12 feature vectors to our data matrix. An example of the six most common colors extracted from an image is shown in Figure 6 below.

Figure 6: The six most common colors extracted from the shown image

We also made use of colorgram's output of color percentages by estimating the size of the Pokemon based on the percentage of each image composed of black pixels. I.e. an image that contained a high percentage of black pixels likely contained a small Pokemon, while an image that contained smaller percentage of black pixels likely contained a larger Pokemon. For each of the images, we recorded this percentage and added it to our data matrix as another feature vector. A visual example of this process is shown in Figure 7 below.

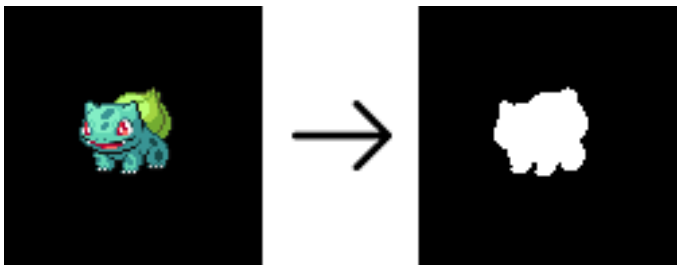

Figure 7: The size of the Pokemon was estimated by the percentage of the image composed of black pixels

After performing these two extraction techniques, we were curious as to whether using general image feature extraction algorithms to detect the "shape" of the Pokemon would improve our accuracy. To test this, we ran our images through MATLAB's implementation of FAST (Features from Accelerated Segment Test) and obtained the most significant coordinates as detected by the algorithm. This algorithm works by iterating through the points in the image and comparing the brightness of that pixel to the ones closest to it (in a Bresenham circle). If the selected pixel is darker than those in a continuous set around it, then it is labeled as a significant feature point [2]. An example of the points selected by FAST is shown in Figure 8 on the next page.

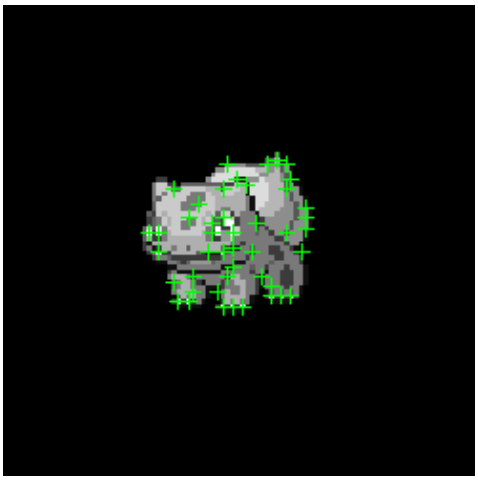

Figure 8: Significant points as detected by the FAST algorithm

We tried adding the (x,y) coordinates of the 20 most significant points as 40 features vectors to our data matrix. Initially, results were promising, as cross-validation suggested a prediction accuracy of up to 72%. However, upon submitting our predicted labels to Kaggle, we found that our accuracy actually decreased compared to simpler methods. We believe that there must have been some set of feature vectors from FAST that was extremely predictive of the training data, but which failed to appear in the test data. Therefore, we abandoned the use of this algorithm and settled for color and size as our additional feature vectors.

7 Model Selection / Comparison of Models Tried

7.1 K-Nearest Neighbors

The first algorithm we experimented with was K-Nearest Neighbors (KNN). Our motivation behind trying this for our first few attempts was twofold: (1) KNN is a simple algorithm that we hoped would work well on our relatively low dimensional data, and (2) KNN should, in theory, be able to pick up on which Pokemon types are similar and systematically distinguish between them.

KNN works by computing the 2-norm distance between a given test point and all of the training points. It then picks the K nearest training point to said test point and has them "vote" on a label. Without PCA or LDA, we were able to achieve an accuracy of 18.85% with K = 10 nearest neighbors. With LDA we were able to achieve an accuracy of 31.9% with K = 15 nearest neighbors While these results were quite acceptable, we hoped to achieve a higher accuracy. Therefore, we moved on to other, more complex, algorithms.

7.2 SVM

The second algorithm we experimented with was Support Vector Machines (SVM) which are typically considered powerful classifiers. SVMs attempt to classify data by creating multidimensional hyperplanes that divide the space in order to try and create fairly homogenous partitions of the data. The classifier then labels data according to whether a data point falls on one side or the other side of the hyperplane. A simple figure that depicts a hyperplane splitting two classes of data is shown in Figure 9 on the next page [3].

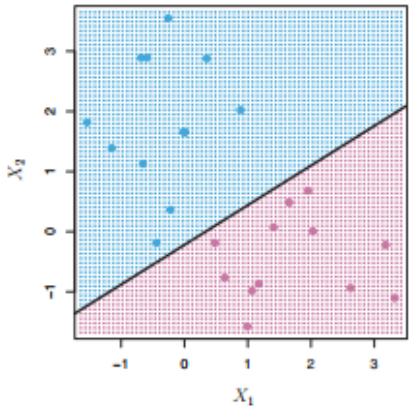

Figure 9: Visual example of the SVM hyperplane. (Source: *An Introduction to Statistical Learning*)

Besides the fact that SVMs were covered extensively in class, we decided to use SVMs because most classification algorithms can be summarized by an attempt to linearly separate data in order to make classifications. One of the attributes of SVMs is the ability to use kernels that allow use to use otherwise computationally expensive non-linear mappings on our data in order to make it linearly separable. These nonlinear mappings make our otherwise inseparable data actually separable. We decided to try the Linear Kernel and Gaussian for our SVMs. The Linear Kernel just gives us back the traditional support vector classifier; however, the Gaussian kernel actually is an infinite dimensional mapping which can result in the SVM fitting the training data so well to the point of overfitting.

We decided to use the sklearn implementation for both the Linear and Gaussian Kernel. Once again, we used LDA to transform our data into lower dimensions for better generalization. For our Linear Kernel, we trained our model with 11 dimensional LDA transformed data and got 32.34% accuracy for our cross validation score. With our Gaussian Kernel, we trained our model with 10 dimensional LDA transformed data and got 32.98% accuracy for our cross validation score. Although the two kernels had different mappings, both of the kernels resulted in similar cross validation scores.

Since SVMs are a complex black box model when in higher dimensions it is difficult for us to see what features our algorithm learned and where classification errors stemmed from. In conclusion, SVM with LDA proved to be a reasonably strong classifier that competes with many of our other methods.

7.3 Random Forest

One of the algorithms that was not discussed in class but which we explored is called Random Forest. Random Forest is an ensemble-based method focused only on decision trees. One reason we decided to try an ensemble learning method is because an ensemble is based on the idea that pooling together multiple weak predictors can form a strong predictor [1].

In order to describe the Random Forest algorithm, we first need to describe a basic decision tree classifier. A decision tree consists of a series of splitting rules that start at the top of the tree that then work its way down to to the terminal nodes of the tree which describe an outcome such as a class. In order to build these trees, implementations use recursive partitioning also known as divide and conquer because it splits the data into subsets which are then repeatedly split into smaller subsets and so on until the implementation determines the subsets of data to be homogeneous enough such as all data in the subset is the same class. A simple example of a decision tree is shown in Figure 10 on the next page [3].

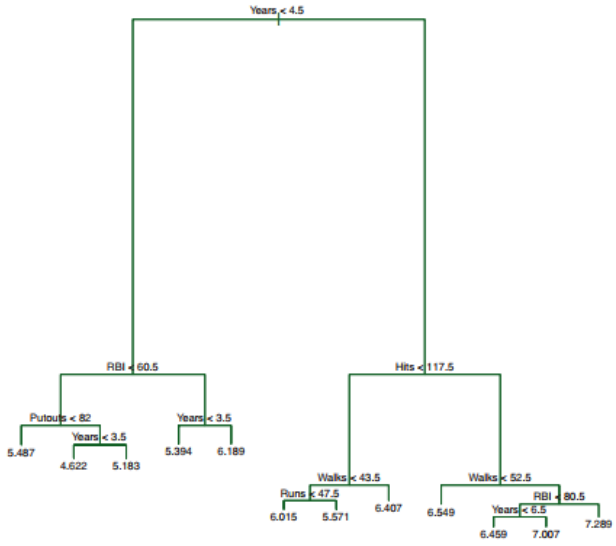

Figure 10: Visual example of decision tree. (Source: *An Introduction to Statistical Learning*)

Another process we need to describe in order fully understand the Random Forest Algorithm is Bagging. Bagging is a technique which generates a number of training data-sets by bootstrap sampling the original sampling data. Bootstrap sampling is a method to obtain distinct datasets by repeatedly sampling from the original dataset with replacement. These datasets are then used to generate a set of models using a single learning algorithm which are then combined using voting for classification.

With Decision Trees and Bagging, we can finally explain Random Forest. Random Forest improves Bagging with Decision trees by decorrelating the trees. In order to do this, Random Forest builds a number of decision trees on bootstrapped training sets; however, when training these trees, a random sample of n predictors is chosen as split candidates from the full set of p predictors. This technique forces the algorithm to not even consider a majority of the predictors at each training step which protects against all the trees using a strong predictor as one of its top splits. Using only a small, random portion of the predictors for training allows Random Forest to overcome potential "curse of dimensionality" issues.

With the benefits of an ensemble and decorrelated trees, we decided to use a Random Forest implementation in sklearn as well as LDA on our data matrix in order to reduce the dimensionality of the data for better performance. For parameter tuning, we found that 24 trained trees and 15 for max depth of the trees produced the best results with a 32.5 % accuracy score in cross validation. However, after observing the predictions of with this model, we found that the model predicted only a few classes with high frequency which lead us to believe the model was weighting certain features and classes too heavily.

Although a common benefit of Decision Trees is their easy interpretability, Random Forest models are not easily interpretable due to them being an ensemble method. Therefore, we can not see exactly what features the algorithm learned and what it typically misclassified. In summary, Random Forest seemed like a promising and interesting machine learning method that had reasonable performance.

7.4 Logistic Regression

For our final model, we used both LDA and logistic regression. Logistic regression uses the logistic function to predict the probability that a given data point is either a 0 or 1. An example of the logistic function with regression coefficients for only one predictor is shown below:

$$p(x) = \frac{e^{b_0 + b_1 * x}}{1 + e^{b_0 + b_1 * x}}$$

The function produces a S-shaped curve where for low balances we predict a probability close to zero and for high balances we predict a probability close to 1. In order to fit the regression coefficients, a maximum likelihood model is typically preferred. The intuition behind maximum likelihood is that we want coefficients that produce a class prediction of a sample which is as close to the actual class of the input sample as possible. An example of a logistic regression classifier is shown in Figure 11 below [3].

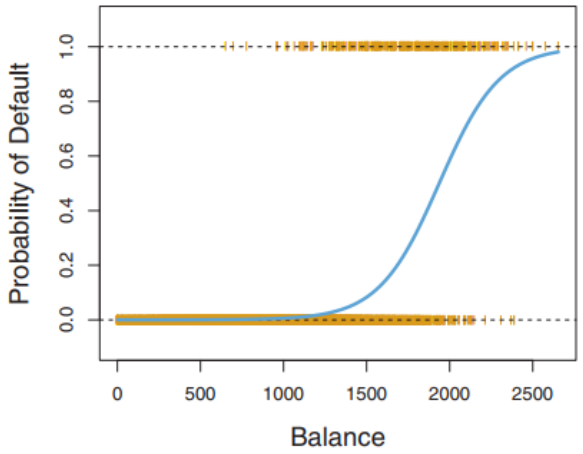

Figure 11: Visual example of logistic regression classifier. (Source: *An Introduction to Statistical Learning*)

Since we are dealing with a multi-class classification problem, logistic regression can be altered to use a one-versus-all scheme in order to do multi-class classification. For example, we can take all fire type Pokemons as positive examples and the non-fire type Pokemons as negative examples. Using logistic regression, we can train a classifier. We then input an unlabeled Pokemon into this fire-type classifier and see the result. The result of this classifier will tell us the probability of the Pokemon being a fire-type. If we repeat the process for the remaining 17 types, we can choose the highest probability from the 18 classifiers and predict that the unknown Pokemon's type is the one corresponding to that classifier.

We decided to use the sklearn implementation of Logistic Regression. Using LDA, we reduced the data to 11 dimensions. For parameter tuning, we used 0.44 as our C which controls regularization of the model. A lower C results in more generalization and less overfitting compared to a higher C value. With C as 0.44 and 13 dimensional data, our best result was 34.08% accuracy in cross validation on our training data, and this model resulted in our best submission on Kaggle with a score of 35.82%. In summary, logistic regression proved to be a strong performer compared to the other algorithms we tested.

8 Methods for Avoiding Overfitting

To avoid overfitting, we used sklearn's cross_val_score function and chose for our cross validation splitting strategy the standard k-fold cross validation where k = 10. This allows the training data to be split into 10 groups. The model is trained on nine of the groups and tested on the remaining group. This is repeated 10 times and the accuracies for those 10 groups are then averaged to provide the cross validation score. This provides a less biased prediction compared to a single split of the training data which can provide an overly optimistic accuracy.

9 Parameter Tuning

To find the best parameter for LDA, we included in our Python code a for loop that would loop through the LDA n_components variable for n = 1:15 and output the best n_components corresponding to the highest cross validation score. n_components is the number of dimensions or columns we want the data matrix to have after dimensionality reduction is performed on it. Shown below in Figure 12 is a plot depicting accuracy vs. value of n_components.

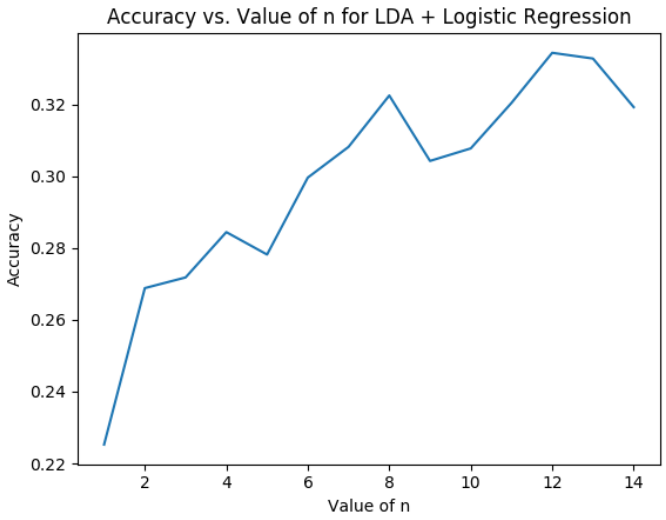

Figure 12: Relationship between the value of n and the accuracy of the model

To find the best C parameter for logistic regression, we wrote similar code to iterate a variable c from 0.01 to 1 with step size of 0.01. For logistic regression, large values of C correspond to less regularization, so as C becomes very large, there is a higher probability of overfitting. With this in mind, we kept C small so the model could generalize well to unseen data and tested values of C between 0.01 and 1. Figure 13 below shows a plot depicting accuracy vs. C. We performed similar parameter tuning on all of our other models.

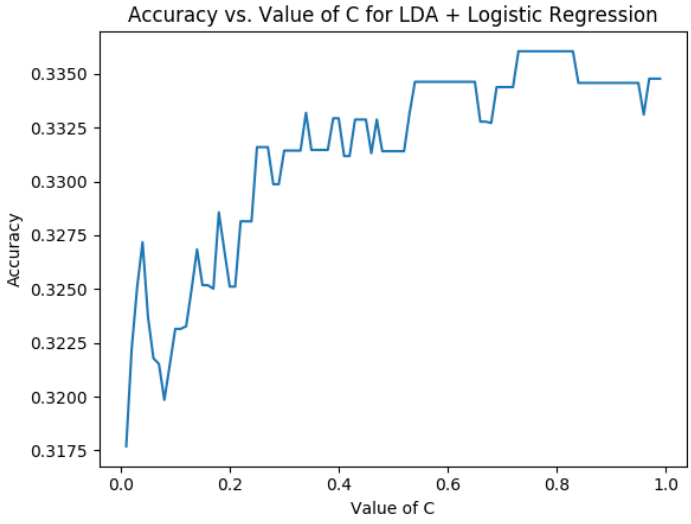

Figure 13: Relationship between the value of C and the accuracy of the model

10 Results and Conclusion

After testing all the previously outlined models, our cross validation scores from training data for various models are shown in Table 1 below. For all of these models, LDA outperformed PCA for the purpose of dimensionality reduction. We found the optimum hyperparameters for the different algorithms by using basic for loops.

Table 1: Results

Model	Dimensionality Reduction	Number of Dimensions	CV Accuracy
KNN	PCA	N = 11	20.46%
KNN	LDA	N = 11	31.9%
SVM (Linear)	PCA	N = 8	25.02%
SVM (Linear)	LDA	N = 11	32.09%
SVM (Gaussian)	PCA	N = 15	20.97%
SVM (Gaussian)	LDA	N = 10	33.3%
Random Forest	PCA	N = 8	21.57%
Random Forest	LDA	N = 9	32.5%
Logistic Regression	PCA	N = 13	25.58%
Logistic Regression	LDA	N = 13	34.08%

In the end, our best model was LDA with Logistic Regression with a submission accuracy of 35.82%. Although the accuracies are nowhere near perfect, we can assume the low accuracies are caused by a small or nonexistent correlation between the stats of the Pokemon and the type of the Pokemon. It seems that the images would be a better indicator for type. If we saw an image of a Pidgeot for the first time and had to guess its type, most of us would immediately guess flying type due to specific features such as wings and its bird-like appearance. Unfortunately, our model was not able to factor in these features and thus resulted in a relatively low score.

Some important things we learned from this project were that overfitting is very possible even with cross validation, how different models have different strengths/weaknesses, how tuning parameters can squeeze out extra percentage points, and how dimensionality reduction can produce better results. Overall, we learned about the fundamentals of the data science process such as exploratory data analysis, feature extraction, and model selection.

11 Code

```
# Import necessary libraries
import numpy as np
import pandas as pd
import glob
import colorgram

# Create color_list and color_list_test which contain top 6 colors for images
color_list = [colorgram.extract(file,6) for file in sorted(\
              glob.glob(r'C:\Users\Robin\Desktop\ELEC301\TrainingImages\*.png'), key =
    lambda name: int(name[46:-4]))]
color_list_test = [colorgram.extract(file,6) for file in sorted(glob.glob(r'C:\Users\Robin\
    Desktop\ELEC301\TestImages\*.png'),
                                   key=lambda name: int(name[42:-4]))]

# 4 sets of RGB values (12 columns) appended to four_colors and four_colors_test (excluding
     black)
# Some Pokemon have less than 5 colors. For these specific Pokemon, the average of the
# first three RGB values (excluding black) is taken and used as the fourth color
four_colors = []
for i in range(0,601):
    if len(color_list[i]) < 5:
        for j in range(1,4):
            four_colors.append([color_list[i][j].rgb.r, color_list[i][j].rgb.g,\
                    color_list[i][j].rgb.b])
        four_colors.append([np.mean([color_list[i][1].rgb.r, color_list[i][2].rgb.r,
    color_list[i][3].rgb.r]), \
                            np.mean([color_list[i][1].rgb.g, color_list[i][2].rgb.g,
    color_list[i][3].rgb.g]),\
                            np.mean([color_list[i][1].rgb.b,color_list[i][2].rgb.b,
    color_list[i][3].rgb.b])])
    else:
        for j in range(1,5):
            four_colors.append([color_list[i][j].rgb.r, color_list[i][j].rgb.g,\
                    color_list[i][j].rgb.b])

four_colors = np.ravel(four_colors)/256
four_colors = np.reshape(four_colors, [601,12])

four_colors_test = []
for i in range(0,201):
    if len(color_list_test[i]) < 5:
        for j in range(1,4):
            four_colors_test.append([color_list_test[i][j].rgb.r, color_list_test[i][j].rgb
    .g,\
                    color_list[i][j].rgb.b])
        four_colors_test.append([np.mean([color_list_test[i][1].rgb.r, color_list_test[i
    ][2].rgb.r, color_list_test[i][3].rgb.r]), \
                            np.mean([color_list_test[i][1].rgb.g, color_list_test[i][2].
    rgb.g, color_list_test[i][3].rgb.g]),\
                            np.mean([color_list_test[i][1].rgb.b,color_list_test[i][2].
    rgb.b, color_list_test[i][3].rgb.b])])
    else:
        for j in range(1,5):
            four_colors_test.append([color_list_test[i][j].rgb.r, color_list_test[i][j].rgb
    .g,\
                    color_list_test[i][j].rgb.b])

four_colors_test = np.ravel(four_colors_test)/256
four_colors_test = np.reshape(four_colors_test, [201,12])

```

```
# Size feature vectors for training and test. Size of the Pokemon is obtained by
    subtracting
# the percentage of black pixels from 1.
size = []
for i in range(0,601):
    prop = 1 - color_list[i][0].proportion
    size.append(prop)
size = np.reshape(size, [601,1])

size_test = []
for i in range(0,201):
    prop = 1 - color_list_test[i][0].proportion
    size_test.append(prop)
size_test = np.reshape(size_test, [201,1])

# X is training data with stats, colors, size and X_unlabeled is test data
dataset = pd.read_csv(r'C:\Users\Robin\Desktop\ELEC301\TrainingMetadata.csv')
unlabeled_dataset = pd.read_csv(r'C:\Users\Robin\Desktop\ELEC301\UnlabeledTestMetadata.csv'
    )
X = np.hstack((dataset.drop(['Type','Number'],axis = 1), four_colors, size))
y = dataset.iloc[:,1].values
X_unlabeled = np.hstack((unlabeled_dataset.drop(["Number"],axis = 1), four_colors_test,
    size_test))

# Remove mean and scale variance to 1 for X and X_unlabeled
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_sc = sc.fit_transform(X)
X_unlabeled = sc.transform(X_unlabeled)

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Iterate over n from 1 to 15 to find best dimension to reduce to with LDA
n_range = range(1,15)
n_scores = []
for n in n_range:
    lda = LDA(n_components = n)
    X = lda.fit_transform(X_sc, y)
    classifier = LogisticRegression(penalty = 'l2', C = 0.72, random_state = 5)
    scores = cross_val_score(classifier, X, y, cv = 10, \
                             scoring = 'accuracy')
    n_scores.append(scores.mean())

print('Best value of n is', np.argmax(n_scores), 'with an accuracy of ', max(n_scores))
plt.figure()
print(n_scores)
plt.plot(n_range, n_scores)
plt.xlabel('Value of n')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Value of n for LDA + Logistic Regression')

# We find best n_components is 12 so we iterate over C from 0.01 to 1 with step
# size of 0.01 for logistic regression
lda = LDA(n_components = 12)
X = lda.fit_transform(X_sc, y)
c_range = np.arange(0.01,1,0.01)
c_scores = []
for c in c_range:
    classifier = LogisticRegression(penalty = 'l2', C = c, random_state = 5)
    scores = cross_val_score(classifier, X, y, cv = 10, \
```

```
                                scoring = 'accuracy')
    c_scores.append(scores.mean())

print(c_scores)

plt.figure()
plt.plot(c_range, c_scores)
plt.xlabel('Value of C')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Value of C for LDA + Logistic Regression')

print('Best value of C is', 0.01*np.argmax(c_scores), 'with an accuracy of ', max(c_scores))


# Apply LDA on X_sc. Apply LDA on X_unlabeled. Learn parameters from the recently
# dimensionality reduced X and untouched y using logistic regression
lda = LDA(n_components = 12)
X = lda.fit_transform(X_sc, y)
X_test = lda.transform(X_unlabeled)
classifier = LogisticRegression(C = 0.45, random_state = 5)
classifier.fit(X, y)

# Best is 33.91% with n = 12, C = 0.42
# Best is 34.08% with n = 12, C = 0.45

# Use parameters learned from above to predict types for X_test, unseen data
y_submission = classifier.predict(X_test)
np.savetxt("yeet3.csv", y_submission, delimiter=",")
```

References

[1] Brett Lantz. *Machine Learning with R, Second Edition*. Packt Publishing, Birmingham, UK, 2015.

[2] Features from accelerated segment test,
`https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test`

[3] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. *An Introduction to Statistical Learning*. Springer Publishing, 2015.