Adiv Padgilwar
COE379L - Joe Stubbs
Project 1

Write Up

In order to feed it into a model, data must be cleaned in such a way that it is as easy and intuitive as possible to process. To make this happen, I split my cleaning process into two separate steps. First, null values. Because the “Unknown” values were part of a categorical column, it was not possible to easily find a median or replacement. Since there were only a few thousand where this was the case, I thought it was suitable to drop the rows that contained that. After this, I also removed all duplicates based on the row ID, assuming it’s acting as a unique key in the dataset. Next, I looked into what columns were actually important. I decided that the name is unique enough to expand the dataset extremely quickly with one-hot encoding, so I dropped it. While it is technically categorical, I believe it does not provide much value in estimating the outcome type as it is more or less arbitrary. Alongside this, I dropped the ID column and the month-year column. Since the month-year described the same thing as the data time, we only needed one, and datetime was much more descriptive and easier to parse. When parsing this data, I opted for a “days since minimum” approach. Thus, each date time value was converted to a float representing days since some value in 2013 (the minimum). The same was done for the Age upon outcome. Keep in mind, this means that rather than looking at the actual age, I am looking more at the relative oldness of the pet since we do not actually know if the pet is alive right now (present time would not work). After the main processing, we are still left with categorical data. Thus, I one-hot encoded all of my non-numeric columns besides outcome type (what we are trying to predict). And finally, I dropped the breed column as it was going to affect the accuracy of our model (given in the instructions).

From the *preparation*, I deepened my understanding of the data columns and how they relate to one another. As for the univariate analysis, I realized a few things. First, most of the categorical variables, like breed or color, spanned a large set of possible values. When I made histograms for them, there were so many labels that the bottom axis was completely blacked out. This was confirmed when I one-hot encoded it and the dataframe expanded to many, many more columns. However, interestingly enough, the color was distributed with a very clear mode. I thought colors would be evenly distributed, but it makes sense that some genes are dominant. I really wonder how this could influence the outcome. Another thing I found was that the outcome type was well distributed. It was not exactly 50/50, so I had to stratify the data, but there were enough data points to ensure the model could be trained on both outcomes effectively. This allowed me to be more confident when creating the training data.

In order to train the model, I had to separate the data into both train and test datasets (train_test_split). While we used a 30/70 split in class, I thought it might be better to use a 10/90 test/train split since there was much more data to work with and I wanted to maximize the amount of data it trained on. Alongside this, I stratified the data to have even distributions of outcome type in both datasets, and then I implemented a random seed. This is just so the results are repeatable. Once I get the data, we can start training the model. I looked at three different

models: KNN, KNN with GSCV, and the SGDClassifier. In order to train the KNN, I picked a somewhat arbitrary n_neighbors value, as it will be later optimized. Then, I fit it on the training data and created the predicted outcome based on x_test. Now for the more interesting one, GSCV. In this one, I made a param grid that allowed us to iterate through a range of possible neighbors. This one also requires a KNN model as an input to keep testing it, so I put in the previous model I trained. Then, I added 5 folds to equally partition the dataset 5 ways. This allowed us to optimize on certain subsets of data and combine the results afterward. Just to make things easier, I also added num_jobs=-1 and verbose=2 to run it on all available cores and print out the progress as it went. This at least let me know it was running and where it might hang. After this, I fit the model and printed the best parameter. As for the SGDClassifier, it works by utilizing the data to make an n-1-dimensional object that allows us to linearly separate the data (given enough of it). This one was a lot easier; I just fit the model based on the training data and predicted the outcome on the test. After this, all three were trained and ready.

I checked the performance by printing out the classification report since it provided all of the relevant values. From that, I determined the KNN with grid search outperforms the normal KNN slightly, but both outperform the SGD significantly. This may be due to the high presence of categorical data and "groupings" that can be naturally formed to predict certain values. These "clumps" are better suited to something distance-related like KNN. Now, the value I looked mainly at was accuracy. I reasoned that in this case, the most important metric is accuracy as we are weighing both adoption and transfer as equally important to the shelter. Predicting what the outcome is allows them to optimize certain strategies for finding foster parents for pets and increase revenue. However, it is also important to look at F1 to make sure it is handling both precision and recall with some grace, as in, not predicting everything as one type. In the case of the SGD, it did exactly that. Overall, I think the models performed – not great. The best I ended up achieving was around 80%. While this is ok and provides some insight, a human could probably reason better. As such, the effectiveness of the model was not very high. As for the SGD, it guessed everything as one class, so I would say that is almost on par with guessing randomly.

As mentioned before, I am not very confident in the results of the model. Due to the amount of data it trained on and the precautions we took in the preprocessing, anything it is super confident in, I think I can also be confident in. However, with anything that is even remotely lower than around an 80% confidence level, I do not know if I would trust it. However, this is all just a guess. In order to truly verify, it would be good to visualize the data and boundary in order to see how it is thinking. From that, we can even visually determine how to optimize neighbors in order to more accurately predict the outcome type. I would also like to check how a random forest performs on the same dataset. If my model works better, I can at least be confident in its superiority above the general baseline.