Machine Learning Engineer Nanodegree

Capstone Project

Brice Yokoyama
August 21th, 2018

Shelter Animal Outcomes

I. Definition

Project Overview

It is estimated that 6.5 million companion animals enter United States animal shelters every year according to the American Society for the Prevention of Cruelty to Animals ("ASPCA"). Approximately 3.3 million are dogs and 3.2 million are cats. It appears that the number is decreasing based on year-over-year estimates done by the ASPCA.

Upwards of 3 million animals are adopted annually, but it is estimated that 1.5 million shelter animals are euthanized each year. The goal of this project is to determine if a confident prediction regarding the outcome of an animal can be determined based on a set of recorded characteristics that have to do with the animal's unique situation.

The dataset is provided by the Austin Animal Center and was hosted by kaggle.com as a "Playground" competition.

The competition homepage can be viewed here:

https://www.kaggle.com/c/shelter-animal-outcomes

All statistics are taken from the ASPCA website:

https://www.aspca.org/animal-homelessness/shelter-intake-and-surrender/pet-statistics

At the outset of this project research was done on multiclass classification methods. There are many classifiers that can be used to approach this problem: SVMs, Naïve-Bayes, k-nearest neighbors, neural networks and decision trees could be implemented as solutions to this problem (https://www.cs.utah.edu/~piyush/teaching/aly05multiclass.pdf). More discussion on this in the "Algorithms and Techniques" section of the project.

Problem Statement

The problem presented by this data is one of maximizing desired outcomes. If it were possible to predict the fate of a shelter animal based on a set of features it would then be possible to focus more resources on animals that need that extra push to get them closer to a desired outcome. The strategy here will be to train various classifiers and then compare them to figure out which is the most trustworthy for this situation. The general procedure will be as follows:

1. Download data and read into dataframe for exploration.
2. Explore the data noting any worthwhile discoveries.
3. Modify data as required for training (Fill NaN, one-hot encoding, etc).
4. Instantiate a classifier and tune it using cross-validation techniques.
5. Repeat for various classifiers.
6. Make predictions on test data for each classifier.
7. Compare the results of the classifiers to the benchmark.

For purposes of this project a random forest classifier has been chosen as the benchmark classifier because it is expected to deliver promising results for this type of problem. Discussion of results will primarily be compared with the benchmark and then the results will be discussed including all classifiers.

Metrics

The results will be determined based on the multi-class log-loss metric. This is the metric used for the Kaggle competition so this project will follow suit. The log loss equation is as follows:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$

Where: N = number of datapoints.
M = number of possible outcomes
y_{ij} = 1 if observation in outcome j and 0 otherwise
p_{ij} = predicted probability that observation I belongs to outcome j

Because this is a multi-class classification problem log-loss is a good choice of metric. Log-loss takes into consideration not just whether or not the prediction is correct, but looks at the probability given by the classifier for the correct outcome. As the prediction probability for the correct classification goes down the log-loss increases. The goal is to minimize the log-loss. I believe this metric makes sense because it will attempt to maximize the accuracy and confidence of model selections. From this article: https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/ it seems to me the only other possible choice it "Classification Accuracy" which is only suitable when there are a similar number of each outcome in the training data. The article also mentions a confusion matrix, which seems more like a way to visualize results rather than an evaluation metric. It might be a good way to gain a

better understanding of the models whereas the log-loss seems better for the actual training of the model; again because it is taking into account not only the prediction made by the model, but the confidence in the prediction.

II. Analysis

Data Exploration

As mentioned in the Project Overview the dataset is provided as part of a Kaggle competition. Two csv files are provided, one training set and one test set. Both the training and test sets contain similar features, except the test set does not contain any information regarding outcomes. The features of the dataset are as follows:

Features:

- AnimalID – a unique ID given to each animal.
- Name – Name of the animal. Value is NaN if no name.
- DateTime – Date and time that the outcome occurs.
- AnimalType – Cat or Dog
- SexuponOutcome – Intact Male, Intact Female, Neutered Male, or Spayed Female
- AgeuponOutcome – Age of animal at time that outcome occurs.
- Breed – Breed of the animal
- Color – Color of the animal

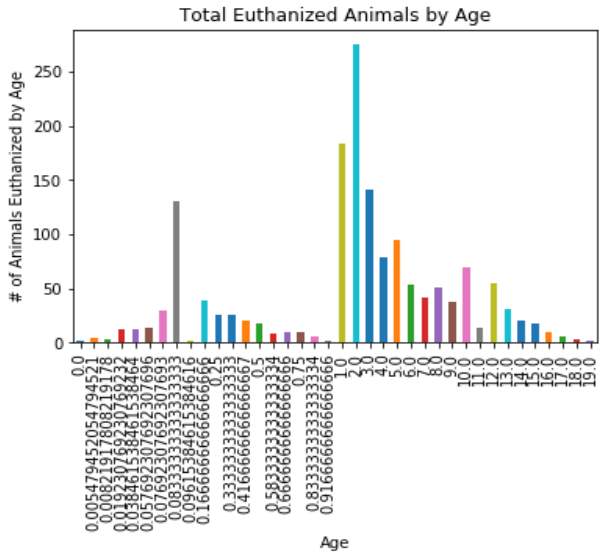
Outcomes:

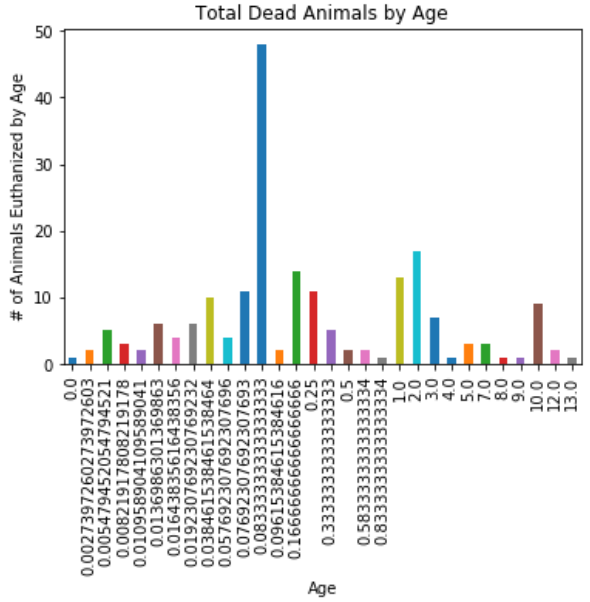
- OutcomeType – Describes the outcome of the animal. Possible values are: Return_to_owner, Euthanasia, Adoption, Transfer, Died.
- OutcomeSubtype – Further describes the OutcomeType.

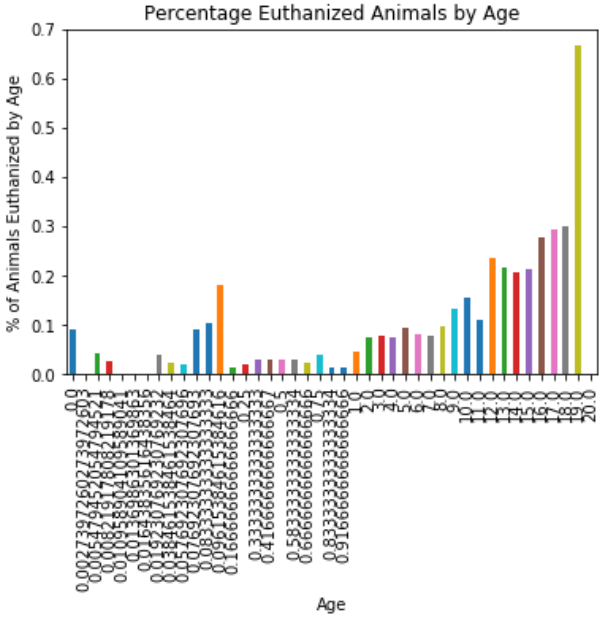
Some basic exploration reveals that the training set contains 26729 animal outcomes – 15595 dog outcomes and 11134 cat outcomes. Out of these instances there are 10769 Adoption, 9422 Transfer, 4786 Return_to_owner, 1555 Euthanasia, and 197 Died. That tallies up to 24977 desirable outcomes and 1752 undesirable outcomes, a 93.4% chance of a desirable outcome. Further exploration reveals: there are 19038 animals with names and 7691 animals that are unnamed. There are 9779 Neutered Male, 8820 Spayed Female, 3525 Intact Male, 3511 Intact Female, and 1093 Unknown; that means 13304 males and 12331 females. Ages of the animals range from 0 years to 20 years. There are 84 unique breeds of dog with 73 distinct color designations while for cats there are 22 unique breeds with 50 distinct colors.

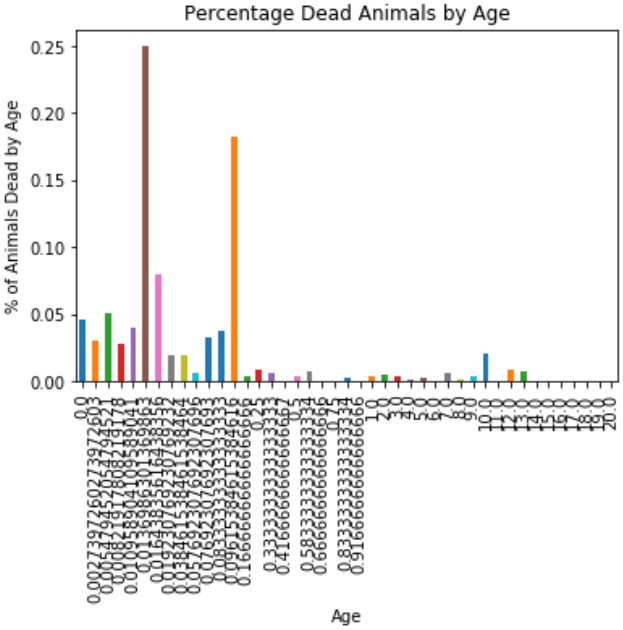
While going through the data some issues become apparent. One problem is that there are too many different names of animals in the dataset, it might be better to modify the data such that each animal is given a value of 0 if unnamed and 1 if named. Another problem is the DateTime column, I believe this should have no bearing on the outcome of the animal since this is not a repeatable value it seems we would gain no insight from this information therefore I plan to remove it from the training data. Splitting the SexuponOutcome data into male/female and neutered or spayed/intact might lead to some insights, so that will be explored. AgeuponOutcome is one of the more problematic columns because the ages can be designated in days, weeks, months, or years. This will be handled by converting all values to years.

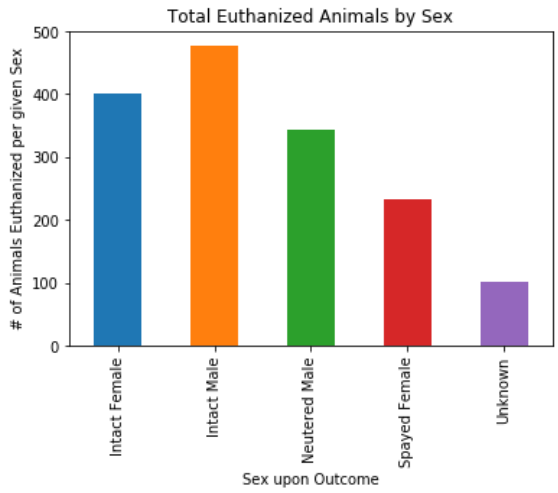
Exploratory Visualization

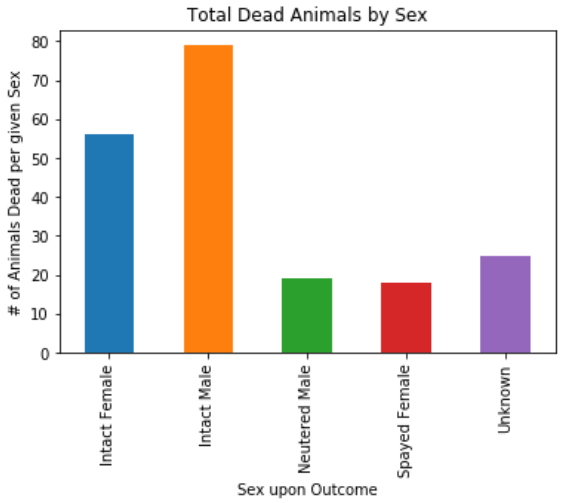

Total Euthanized Animals by Age
# of Animals Euthanized by Age
Age

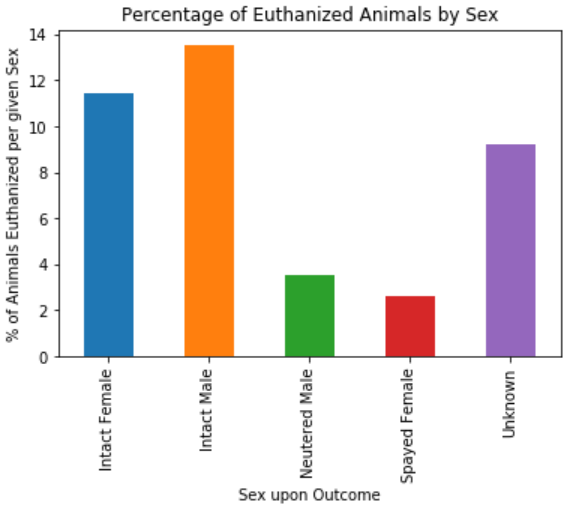

Total Dead Animals by Age
# of Animals Euthanized by Age
Age

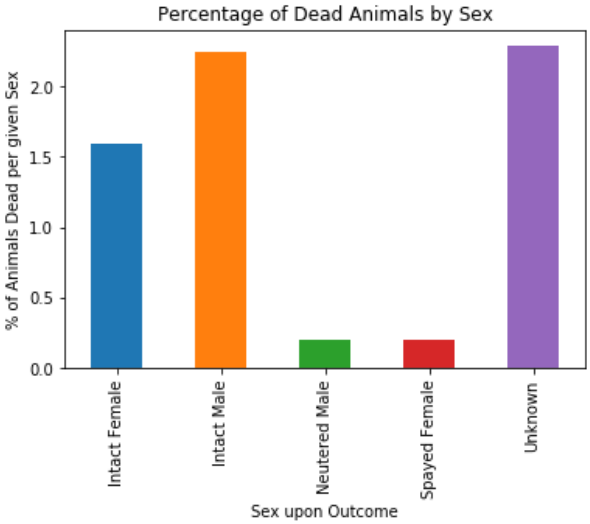

Percentage Euthanized Animals by Age
% of Animals Euthanized by Age
Age


Percentage Dead Animals by Age
% of Animals Dead by Age
Age









The above visualization are displayed because they identify some trends in the data and also show how we can avoid misreading the data. At the top of the previous page are plots of Total Euthanized Animals and Total Dead Animals; below on the above page are plots showing the percentage of animals in their respective age group that were Euthanized or Dead. Similar plots are displayed on this page but with the animals grouped by their SexUponOutcome feature. The first thing to notice is that the plots showing totals of outcomes and the plots showing percentages of outcomes show different trends. For instance, comparing the Total Euthanized Animals by Age plot with the Percentage of Euthanized Animals by Age: it can be seen that while there are more total animals euthanized that are around 2 years old it actually becomes more likely that animals will be euthanized as they get older. The trend is better represented by the percentages rather than the total because the data may simply have more data from any given feature.

Algorithms and Techniques

The idea here is to use a few different classifiers to see which one performs the best. In addition to the benchmark classifier, which will be a random forest classifier, adaboost, gradient boosting will be compared. These classifiers were chosen based on articles found regarding kaggle competition results (https://www.quora.com/What-percent-of-Kaggle-competition-winners-ultimately-use-random-forests-in-their-solutions).

The benchmark will be the random forest classifier. This classifier uses a parallel ensemble method where multiple decision trees are averaged in order to come to a value (also known as aggregation). Each decision tree in the forest uses a random subset of the training data with replacement, meaning the same value can be used to train multiple decision trees (also known as bootstrapping). Also, each decision tree is trained on a random subset of the feature set, this allows for more diversity amongst the forest of trees. The result is many uncorrelated trees which decreases the variance through averaging.

One of the classifiers tested in this project is adaboost, short for adaptive boosting. This classifier combines multiple weak learners to create a strong learner, this method is known as boosting. Boosting reduces error by reducing bias and variance by aggregating the output from many models. Adaboost is a sequential ensemble method in that it uses the results from the previous weak learner to influence the next weak learner's results. Specifically, data points that are misclassified by the first weak learner are given higher weights to be considered by the next weak learner. Weak learners are then given a weight based on their accuracy so more accurate weak learners influence the final value more.

The other classifier testing in this project is a gradient boosting classifier. As can be seen from the name it is a boosting classifier similar to adaboost, meaning it combines many weak learners to form a strong learner. In this case the classifier uses a gradient-descent like algorithm to minimize error. Specifically, gradient boosting involves sequentially adding weak learners that account for the error in the previous weak learners. The final value is then calculated by the various weighted weak learners.

Most of the features will be one-hot encoded to be made usable by the classifier. The only features that needn't be one-hot encoded are the AgeuponOutcome feature, which will have to be modified in another way to be useful, and the Name feature, which will be simplified to "HasName" and given a value of 1 if the Name feature is not NaN. The AgeuponOutcome has values based on days, weeks, months, and years. These values are mapped to their equivalent value in years in a new feature column AgeinYears.

Sources:
https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/,
https://blog.statsbot.co/ensemble-learning-d1dcd548e936,
https://medium.com/machine-learning-101/https-medium-com-savanpatel-chapter-6-adaboost-classifier-b945f330af06,

https://hackernoon.com/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38c,
https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/,
https://www.youtube.com/watch?v=sRktKszFmSk

Benchmark

The benchmark model will be a random forest classifier. The random forest classifier has a good track record when used for multi-class classification. After preprocessing and training on an "out-of-the-box" (only random_state is set to 0) random forest classifier a score of 2.61904 using the log-loss formula is obtained.

III. Methodology

Data Preprocessing

There are many preprocessing steps that need to take place to get the data to a point where it is useful to the classifiers. As discussed in the Data Exploration section some of the modifications are as follow:

- Convert "AgeuponOutcome" feature so that all values are on the same scale (days, months, years). In this case years is used. All unique values present in the feature are mapped to an appropriate value in years and a new column "AgeInYears" is created to store the values.
- Create new column "HasName". Values of 0 are assigned to animals who do not have names and values of 1 are assigned to animals who have names.
- Drop the "DateTime" feature as this seems to be completely arbitrary.
- One-hot encode the "AnimalType" and "SexuponOutcome" categories.
- Drop the "Breed" and "Color" categories. As there are many variations of breed and color that I think would lead the classifier to over fit.

For the results data the "OutcomeSubtype" is dropped and the "OutcomeType" is one-hot encoded.

The data is then split into results and features for the classifiers to be trained on. Up to this point they were in one dataframe.

Implementation

For each of the classifiers the following steps are taken:

1. Classifier is initialized with all parameters set to default and a random_state = 0 for reproducibility.
2. Classifier is trained on training data.
3. Classifier predicts outcomes for test data.
4. Training features are dropped based on the feature_importances_ parameter of the classifier. Only features with importance greater than 0.10 are kept.
5. Classifier is trained on modified training data.
6. Classifier predicts outcomes for modified test data (based on modified training data).
7. Parameter matrix is initialized based on classifier parameters. Parameters are as follows:
    a. Random Forest: n_estimators, max_depth, min_samples_leaf
    b. Adaboost: n_estimators, learning_rate
    c. Gradient Boosting: n_estimators, max_depth, min_samples_split, min_samples_leaf, subsample, max_features, max_leaf_nodes
8. Gridsearch is implemented to optimize the parameter matrix using the sklearn neg_log_loss as a scoring metric.
9. Steps 7 and 8 are repeated until the parameter range is narrowed down sufficiently. Initially parameter ranges are very wide.
10. Classifier predicts outcomes for test data.
11. Steps 7, 8, & 9 are repeated with the modified training set from step 4.
12. Classifier predicts outcomes for modified test data from step 6.
13. The results from each classifier are modified to meet the kaggle competition's submittal requirements (column names and order) and output to a csv file. The csv file is submitted to the kaggle competition page and a score is given.

Refinement

As stated in the last section each of the classifiers is run once with only default parameters and then a gridsearch is utilized in an attempt to optimize the model.

SCORES	No Tuning	GridSearch	Feature Importance	GridSearch w/ Feature Importance
Random Forest	**2.61904**	**3.33648**	**2.66849**	**3.44172**
Adaboost	**1.60682**	**2.43352**	**1.61102**	**2.60159**
GradientBoosting	**3.32590**	**3.25721**	**3.16212**	**3.03811**

In many instances the "out-of-the-box" model performed better than the "optimized" model. The same is true for both gridsearch and cutting down the training sets to only include important features. See previous section for hyperparameters tuned.

IV. Results

Model Evaluation and Validation

The final model is an adaboost classifier with default parameters (n_esitmators = 50 and learning_rate = 1). This proved to be a better model than the "optimized" adaboost classifier using a gridsearch. I would say that the model is not aligning with solution expectations. I was convinced at the start of this project that the scores would be much better and that the optimized model would perform much better, instead it is performing slightly worse. To check the robustness of the model I removed a quarter of the training data and checked the score once again. The classifier trained on less data produced a score of 1.60787 compared to the 1.60682 of the classifier trained on the full set of training data. From this we can see that the difference is minor. It still seems that the results from the model cannot be trusted since the log-loss score is so high.

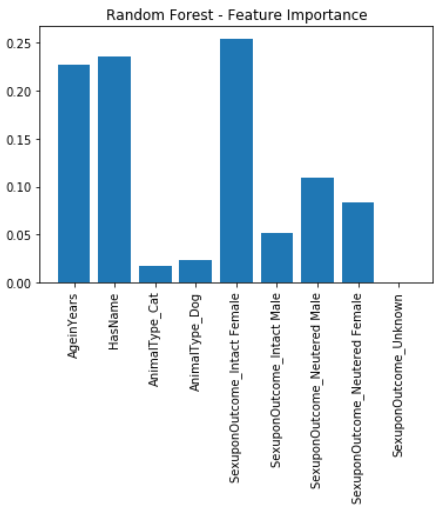
Justification

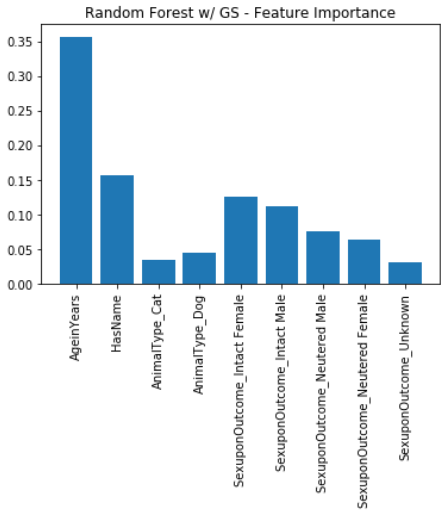
I believe the difference is clear when comparing the models I've chosen for this project. The benchmark random forest classifier performs better than the gradient boosting classifier, but does not outperform the adaboost classifier. The best score achieved by any variation of the random forest classifier was 2.61904. The best score achieved by any variation of the gradient boosting method was 3.03811 and the best score over all by any of the classifiers was the adaboost with a score of 1.60682.

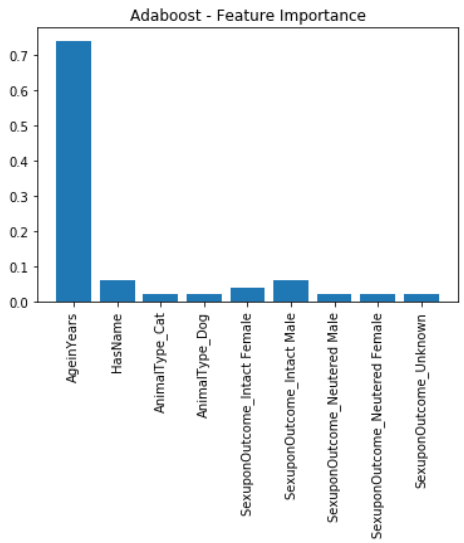
The classifier does not perform well overall, but it is the best of the three I've tested for this project.

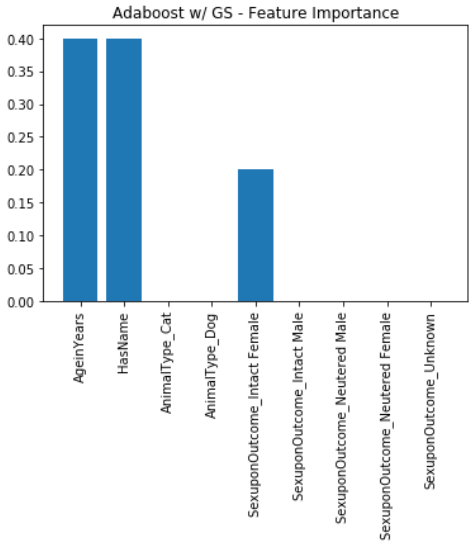
V. Conclusion

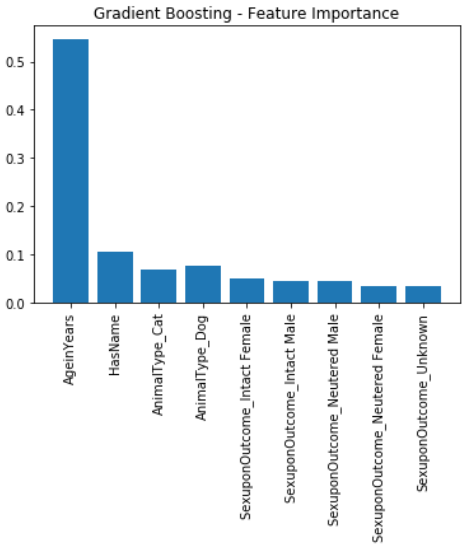
Free-Form Visualization

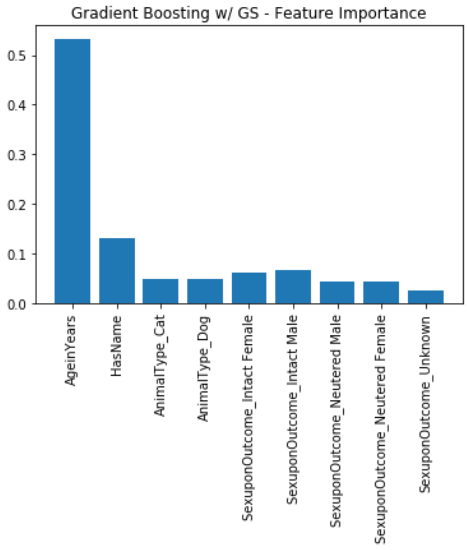












Visualized above are the feature importance values for each of the classifiers that isn't trained on the modified training data. One thing to notice is that the feature importance changes significantly between the random forest classifier and the random forest classifier tuned with gridsearch. The same can be said for adaboost. I suppose this is saying that the feature importance is dependent on the parameters of the classifier.

Reflection

This capstone was quite daunting from the start. From picking a project to starting and carrying it through to some kind of completion didn't seem possible at some points. I'm glad I was able to get through it and I've definitely come away with a great appreciation for what it takes to go through a project. I believe the dataset I was working with was a good one for a beginner to work with. There were not too many features to look at and there was some intuition that could be used to get a feel for the data (maybe that's not a good thing when we're trying to be more

technical about the data, huh). In any case, exploring the data and going through all the dataframe manipulation was a rewarding experience. As soon as I got a grasp of how to manipulate the dataframes (thanks to kaggle tutorials!) I was excited to try out different combinations of features and compare them next to each other in various graphs. I think I gained the most out of this project while exploring the data and getting to know dataframes a little bit better.

Running the classifiers turned out to be surprisingly simple, but tuning the parameters was completely baffling to me. I was not able to produce expected behavior using the gridsearch method unfortunately. I am really not sure what the reason for this is. Another thing that I haven't figure out is why the feature importance changes based on the parameters of the model. As seen in the freeform visualization the feature importance changes between the tuned and un-tuned model. That is not very intuitive to me because I would have thought that the feature importance is independent of the models.

Improvement

I think there are quite a few things I could improve upon in this project. One would be the overall organization of the project. When I was starting out I went a bit crazy with the graphs and had no labels or nice naming convention. It was really free-form and chaotic. I think it would have been a better idea to brainstorm on paper and come up with a methodology that I could walk through in steps and have that show in my jupyter notebook. It would be much easier to follow and would cut down on additional work if I were to make modifications. Most of what I'm talking about is the modified data that I used for visualizations.

Another area that could use improvement is the implementation of the classifiers. I'm not 100% sure I was implementing the gridsearch in the most efficient way. After some reading I discovered that the order of the parameters in the parameter matrix DOES matter and could affect the results. Because of this I believe the results could be better in this project. I also was not sure which parameters I should be optimizing through gridsearch and which parameters I might be able to guess at using some kind of rule-of-thumb. This made the process of gridsearching very slow and painful.

I believe the handling of the training features could be handled differently as well. I discarded entire columns of data because I was not sure how to get them to work for me. Breed and color may have unlocked additional correlations that the models could have picked up on and used to score much better. This is one of the things I would like to get better at.