|  |  |
| --- | --- |
| Machine Learning Engineer Nanodegree | Rafael Torres |
| **Capstone Project** | January 16, 2017 |

# Definition

## Project Overview

Animal shelters across the United States end up admitting a total of 7-to-8 million new animals each year. These shelters are often able to find new homes for their animals; however, about 35% of them end up being euthanized, as these shelters are not able to find new caregivers for them[[1]](#footnote-1). Utilizing historical data, and Machine Learning (ML) techniques, it could be possible to predict animal outcomes based on certain features, which could help these shelters to refocus their budgets and efforts to help the most needed segments of their animal population in finding new homes.

This project was inspired by the [Shelter Animal Outcomes project](https://www.kaggle.com/c/shelter-animal-outcomes) featured in [Kaggle](http://www.kaggle.com).

## Problem Statement

In the US, more than 2.5 million animals per year end up being euthanized in animal shelters, because they are not able to find new homes for them. The [Austin Animal Center](http://www.austintexas.gov/department/aac) has made publicly available relevant historical information pertaining dogs and cats they have handled over a period of almost 3 years.

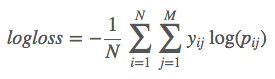
The goal of this project is to utilize supervised learning techniques to define a model to help the shelter to predict the outcome of their animals, so that they can in turn identify segments of their animal population that need extra help in finding new homes. The tasks involved are:

* Read and preprocess the historical data provided by the shelter.
* Split the data into train and test datasets.
* Train the following supervised learning classifiers with the train dataset: Decision Trees, SVM, XGBoost.
* Test the models with the test dataset, using Log Loss.
* Compare the results and select a model for hyper-parameter refinements.
* Use a manual GridSearchCV-approach to iteratively refine the selected model, while re-calaculating the Log Loss.
* Corroborate the results of the optimized model via Kaggle’s evaluation engine.

## Metrics

[Kaggle’s evaluation engine](https://www.kaggle.com/c/shelter-animal-outcomes/details/evaluation) was leveraged for this project. The platform uses Logarithmic Loss (aka Log Loss) to evaluate the model’s performance. It was asked by the evaluation engine to submit predictions as a probabilistic distribution of the possible outcomes. Given such conditions, Log Loss was a natural metric to be used in this case.

Log Loss is defined as “the logarithm of the likelihood function for a Bernoulli random distribution.”[[2]](#footnote-2) The function is defined by:



Equation 1: Log Loss

*Where:*

* *N = number of examples*
* *M = number of classes*
* *yij = binary variable indicating whether class j was correct for sample i*

In plain terms, Log Loss works by penalizing wrong predictions, and the penalization is even more severe for more “wrongly” confident predictions; i.e., the more the predicted probability diverges from the actual value, the more the Log Loss increases. All of this means that the lowest (closest to zero) the Log Loss score for a model, the better. Fortunately, Log Loss is also available in Python’s [sklearn](http://scikit-learn.org/) via the [sklearn.metrics.log\_loss](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html) function.

For this project, local Log Loss scores were calculated as the first step using sklearn, and subsequently evaluated with Kaggle’s engine for comparison with other previously submitted models/scores in their [public leaderboard](https://www.kaggle.com/c/shelter-animal-outcomes/leaderboard).

# Analysis

## Data Exploration

The data utilized in this project is available here:

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

There are 2 main files included in that link:

* *train.csv*
* *test.csv*

### The train.csv file

This file contains the dataset used for (a) training the ML algorithms, and for (b) testing the algorithm locally. The data set size is 26,729 x 10 (rows x cols). The columns are labeled as such:

* AnimalID
* Name
* DateTime
* OutcomeType
* OutcomeSubtype
* AnimalType
* SexuponOutcome
* AgeuponOutcome
* Breed
* Color

Here’s a summary of the shape of the dataset:

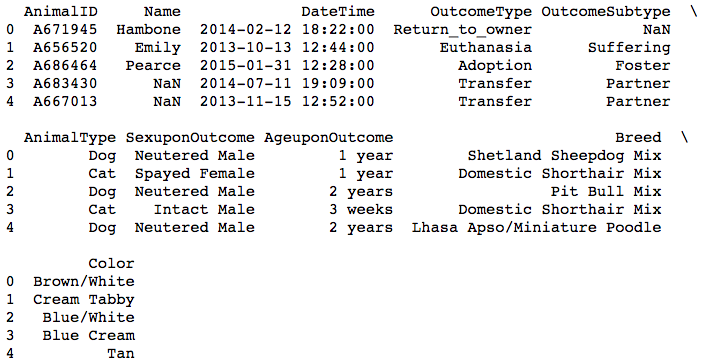


Table 1: Panda's DataFrame head of train dataset

And here’s a broad description of the dataset:

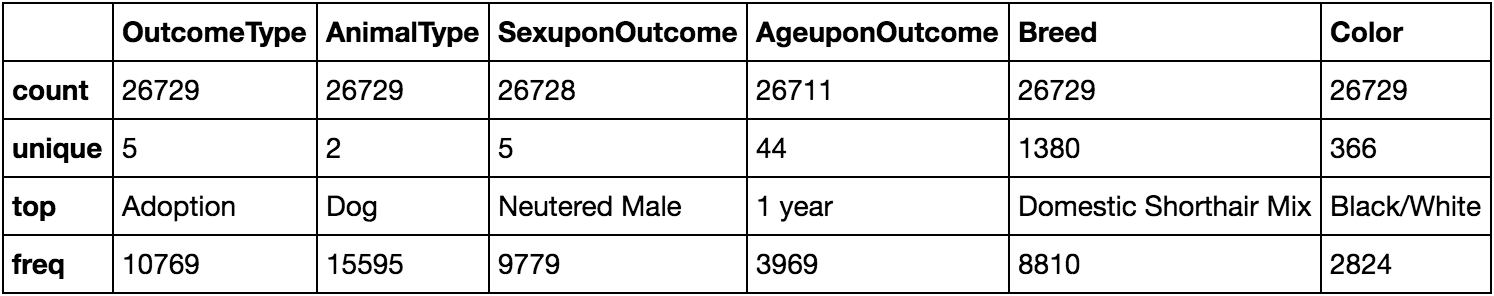


Table 2: Train dataset description

The data represents the shelter’s dogs and cats population, with their corresponding outcome, as recorded from October 2012 through March 2016.

The target label for this project is OutcomeType. The ultimate goal we’re mostly interested is minimizing the number of animals that are euthanized, or die in the shelter. For the purposes of this study, the OutcomeSubtype is irrelevant. Similarly, AnimalID, Name and DateTime don’t give any insight into trying to predict animals’ outcome – these are data points useful for internal tracking purposes only. For that reason, those data points were ignored as well.

In short, the features and target for our ML algorithms are:

* Features: AnimalType, SexuponOutcome, AgeuponOutcome, Breed, Color.
* Target: OutcomeType.

Upon exploring the data, it was quickly observed that most of the features have a very high variety of “categorical” values. For example, Breed has values like “Abyssinian Mix”, “Affenpinscher Mix”, and many other ones. Here’s a summary of the variation of unique values (the number of unique value) shown per feature:

* AnimalType: 2
* SexuponOutcome: 5
* AgeuponOutcome: 176
* Breed: 5,520
* Color: 1,464

The train dataset is made up of approximately 58% dogs and 42% cats. About 6.5% of the population in the dataset was recorded as “Euthanasia” or “Died”, which amounts to 1,752 animals.

The following table shows the group-by OutcomeType breakdown of the data:

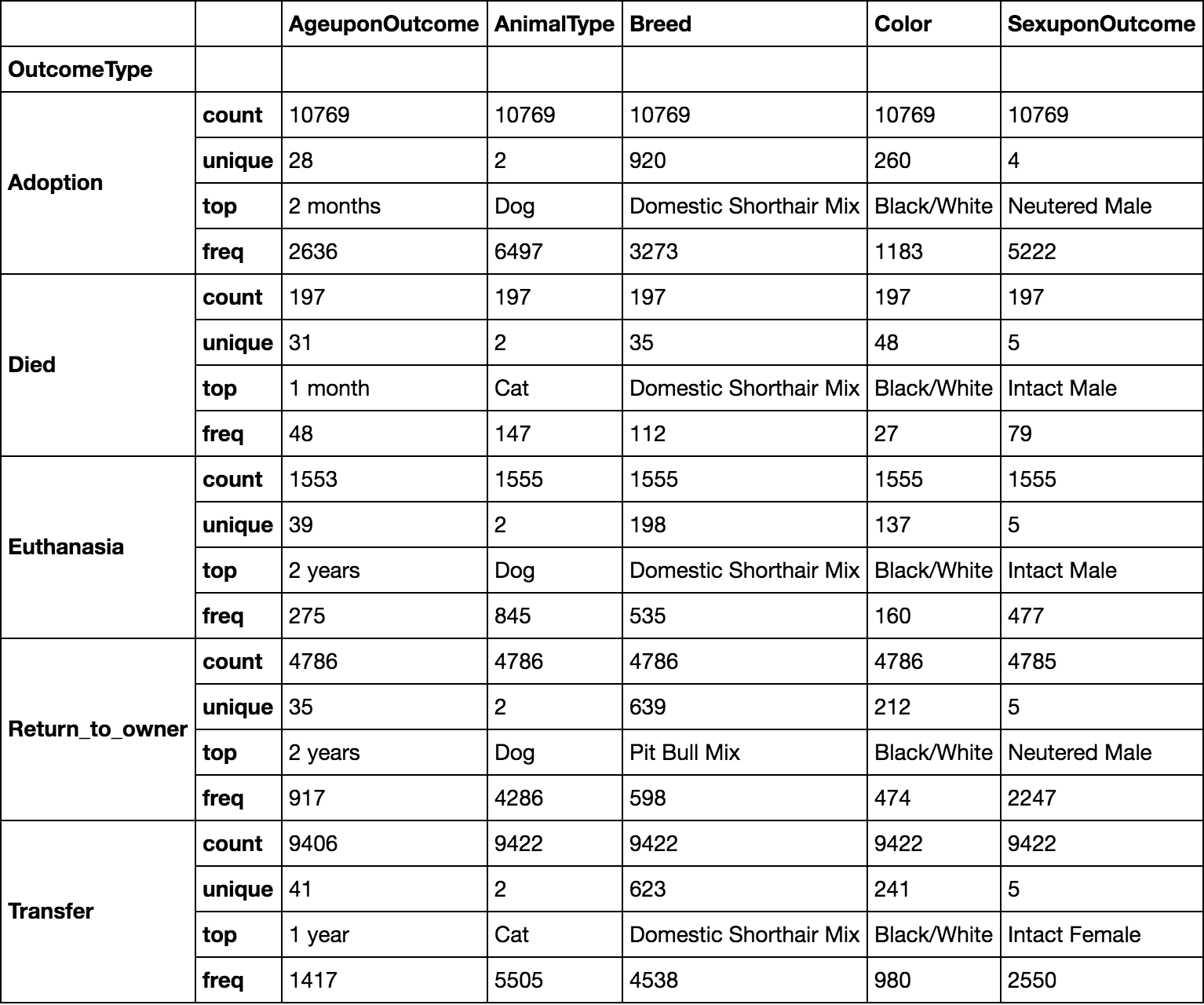


Table 3: OutcomeType groups descriptions

It’s interesting to note that the majority of the animals euthanized were dogs. Also, the age of the population most euthanized was 2 years old. On the other hand, it appears that the category of animals mostly recorded as “Died” were cats, while the age for that outcome is usually 1 month old.

As for the target, the value that we are interested in predicting, fortunately only 5 unique values are part of it:

“Adoption”, “Died”, “Euthanasia”, “Return\_to\_owner”, “Transfer”.

The large variation of values for the data features represented a challenge. Moreover, the amount and value of unique variables between the *train.csv* and the *test.csv* files differed, which made things a bit more challenging. The approach used to deal with these factors is explained in detail in the [Data Preprocessing](#_Data_Preprocessing) section of this document.

### The test.csv file

This file contains the dataset used to make the predictions for a formal submission to the Kaggle evaluation engine. This dataset size is 11,456 x 8 (rows x cols). The columns are labeled as such:

* ID
* Name
* DateTime
* AnimalType
* SexuponOutcome
* AgeuponOutcome
* Breed
* Color

Similarly, with this dataset the same column labels as the ones considered for the train dataset were taken into account.

## Exploratory Visualization

Dissecting the data while focusing on the least desirable outcomes (“Euthanasia” and “Died”), the correlation of Age (in days) and the count of these outcomes shows the following:

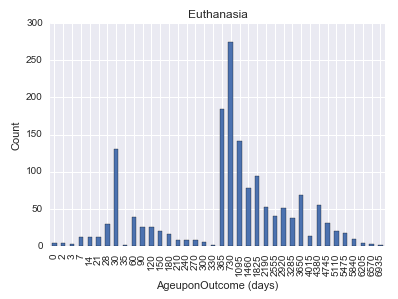


Figure 1: “Euthanasia” OutcomeType count vs AgeuponOutcome Bar Graph

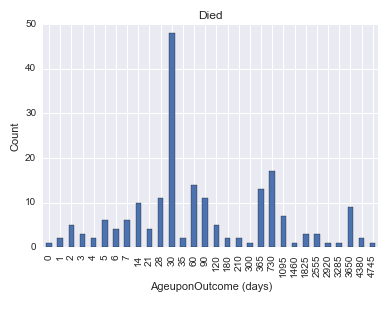


Figure 2: “Died” OutcomeType count vs AgeuponOutcome Bar Graph

As it can be observed in Figure 1, the peak of events happens at around the 2-year mark of the animal’s age (730 days) and at the 1-year mark (365 days). The count is still high at the 3-year mark (1095 days), after which point it starts to become lower. Another interesting time mark is at 30 days, in which the count is about 125, almost at the same level as at the 3-year mark. This seems to challenge the intuition that euthanasia would only happen to older animal population. It would be interesting to find out what’s causing such event at an early age, and see how that information can be used to improve model predictions[[3]](#footnote-3).

As for Figure 2, the peak count happens at 30 days (which concurs with the finding made on the first graph at the same animal age). Similarly, the “Died” outcome count also goes up around the 2- and 3-year marks.

This suggest that the first-month mark (30 days) is an important event for the shelter’s animals, as it can have certain significance on determining possible non-desirable outcomes for them. The same implication could be done towards animals reaching an age of 2-3 years.

## Algorithms and Techniques

The analysis made in this project relied on python/sklearn. The approach utilized in this analysis can be described in this fashion:

1. Data exploration: to better understand the data.
2. Data processing: to normalize the data, as many features were originally not quantifiable. Also a split of the train data was obtained (at 80% train vs 20% test data split) for the models’ training/testing, using sklearn.model\_selection.train\_test\_split.
3. ML algorithm training: see below for the list of algorithms selected.
4. ML testing: using sklearn’s metric, the Log Loss for each model was calculated. Additionally, submissions to Kaggle’s evaluation engine were made, in order to compare models’ outcomes.
5. Model tuning/refinement: from the previous model comparison, the most promising one was selected for further optimizations.
6. Re-calculation of Log Loss: to measure the delta of the optimized model and determining whether any improvement was attained.

Since the end goal is to predict the OutcomeType of the animals, the analysis was focused on different classification algorithms. Three were selected to be used in this project:

* Decision Trees: Due to the simplicity of this algorithm, it is usually a good starting point to obtain some base-line predictions. Decision trees work by creating a tree-like structure to decide the outcome of the prediction based on values of the data features. In simple terms, it can be thought of asking yes/no questions for whether certain features have particular values, and making a prediction based on the answers.
* SVM: Based on personal experience, SVMs have always delivered a higher level of accuracy, albeit at much slower pace due to the higher complexity of the algorithm. For this reason, this was one of the algorithms chosen. At the end, a better performance was obtained with SVM, but its slowness made it practically impossible to fine-tune. SVMs work by creating linear “hyperplanes” (or decision boundaries) among the outcome classes as represented spatially in an n-dimension graph (where n is the number of features)[[4]](#footnote-4). One advantage of SVMs is that they ensure hyperplanes have the maximum possible margin between the nearest samples of the adjacent classes, which enhances the margins for error in predictions. Moreover, for more complex problem spaces (with non-linear decision boundaries), SVMs have the additional advantage of being able to use the [Kernel trick](https://en.wikipedia.org/wiki/Kernel_method#Mathematics:_the_kernel_trick)[[5]](#footnote-5).
* XGBoost: In an attempt to combat the inefficiency of SVM, with the hope to not only get faster results but also higher performance, it was decided to give XGBoost a try. This is a gradient boosting algorithm that uses advanced optimization techniques to improve efficiency dramatically[[6]](#footnote-6). The method used by XGBoost can be thought of creating multiple simple/shallow decision tree prediction models (aka weak learners) and combining their outcomes to produce a more robust model (aka strong learner). At the end, it turned out that both the accuracy and efficiency of this algorithm were far above the previous 2 for our dataset.

## Benchmark

Since this project is inspired by the Shelter Animal Outcomes competition in Kaggle, originally its public leaderboard was used to get an idea of the realistic benchmark numbers. However, as it turns out there’s a bug in the competition that, when exploited, it allows submitters to get near-perfect (or even perfect) Log Loss scores[[7]](#footnote-7).

With that in mind, looking at submissions the numbers seem to vary from all the way up at >34 points to near 0 points. The sentiment by browsing through the [competition’s forum](https://www.kaggle.com/c/shelter-animal-outcomes/forums) is that scores below 1.0 are attainable without exploiting the bug.

Our benchmark for this project was to get below the 1.0 score. At first, a base-line score was established by using a Decision Tree classifier, and consequently look for improvements by using the other 2 models and applying optimization techniques.

# Methodology

## Data Preprocessing

As previously explained in the [Data Exploration](#_Data_Exploration) section of this document, both the train and the test datasets contain non-quantifiable (string-value based) features that needed to be processed before being able to be consumed by the ML algorithm.

For this, initially the one-hot-encoding approach (aka dummy variables) was used; this, however, ended up generating close to 2000 columns/features in the post-processed DataFrame. This made it a bit cumbersome to consume. Alternatively, a better approach was to use sklearn’s [LabelEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html), which would result in the same amount of columns/features as in the original datasets, except for the AgeuponOutcome, which required a distinct approach (as explained below).

The following features were processed, using the normalize\_data function documented below:

* AgeuponOutcome: The original dataset contains string entries on this feature as such: “1 day”, “2 weeks”, “3 months”, “4 years”. These data values were converted to an int, representing “days” (the lower common denominator).
* AnimalType: 2 unique values.
* SexuponOutcome: 5 unique values.
* Breed: 1,678 unique values.
* Color: 411 unique values.
* OutcomeType: 5 unique values.

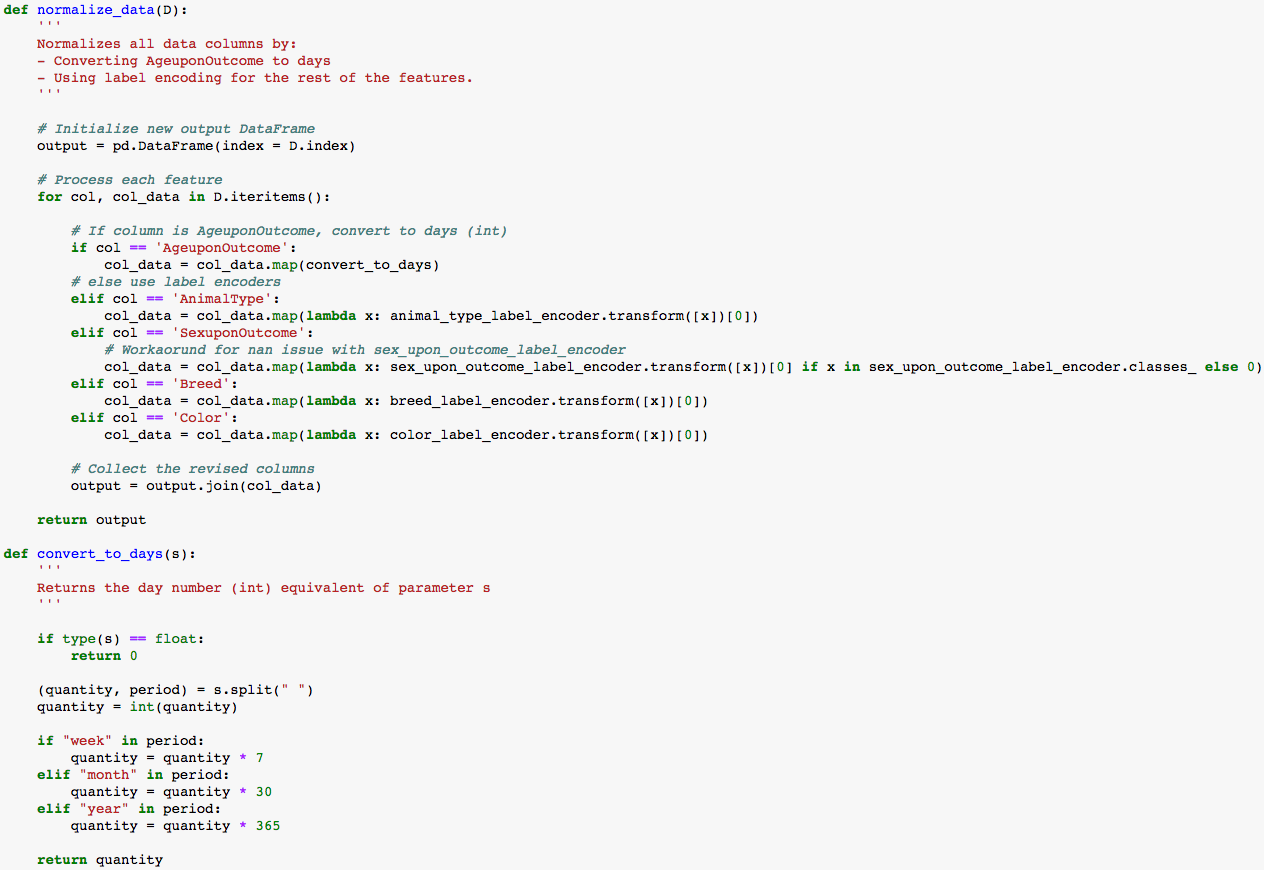


Figure 3: Functions used to normalize the data

This preprocessing step was necessary for both datasets (the one in *train.csv* as well as the one in *test.csv*). Given that some feature values were unique to one of the datasets, the label encoders for each feature were created based on the combination of possible values for the same feature in both datasets. This allowed for using the same label encoding values across the board – an approach necessary to maintain the model’s integrity through training and testing.

Similarly, another preprocessing step that was necessary before being able to submit predictions to Kaggle’s evaluation engine, was to convert the predictions to a [predefined csv format](https://www.kaggle.com/c/shelter-animal-outcomes/details/evaluation) which would include the animal IDs to the predicted records. This processing was done via a custom python function that leveraged the power of [numpy](http://www.numpy.org/) to manipulate data arrays and export them to a csv file, as shown below:

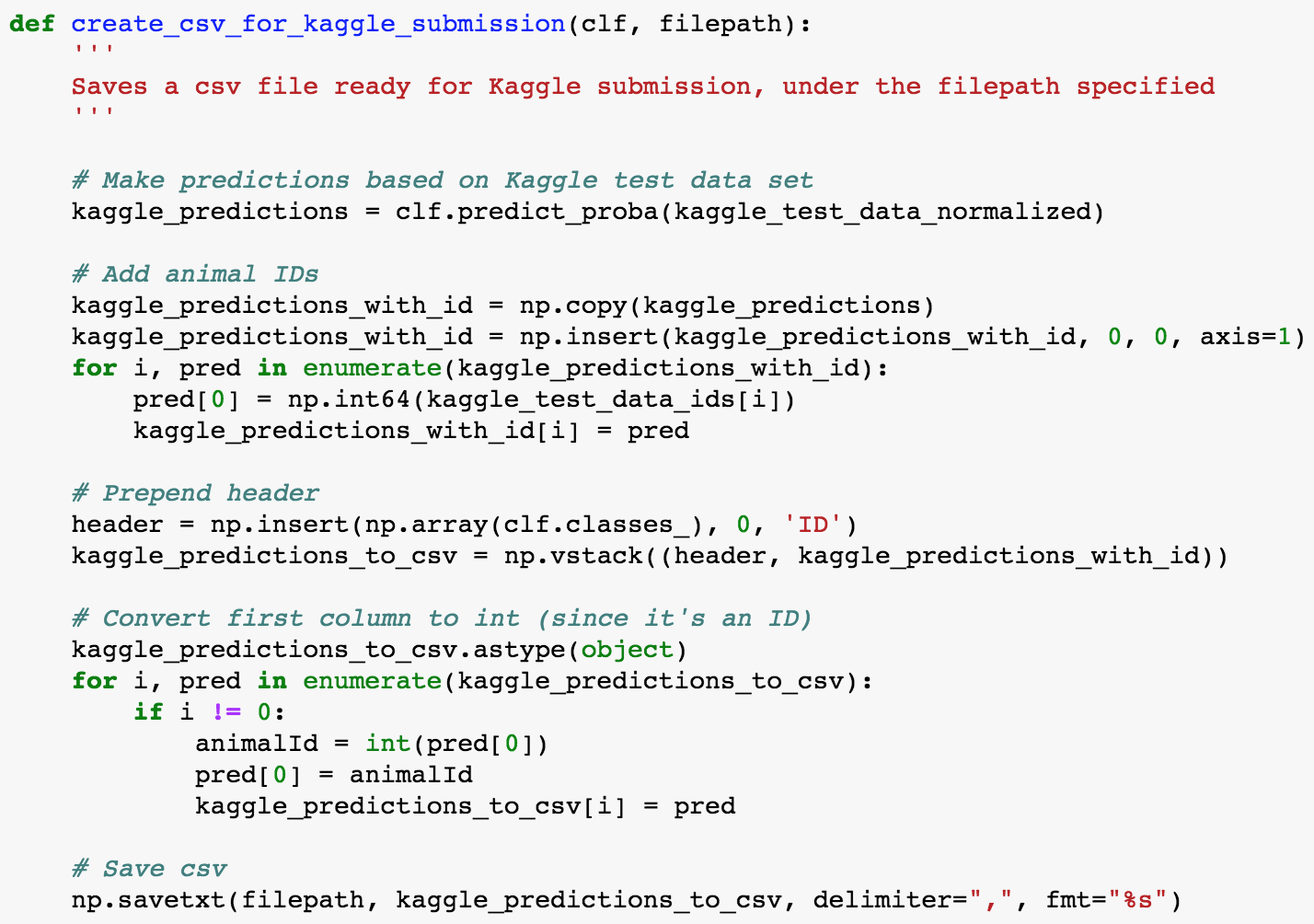


Figure 4: Function used to prepare prediction submissions to Kaggle’s evaluation engine

## Implementation

After pre-processing the data via the method described above, the data in *train.csv* was split in 80/20 (train/test data). Using the 80% train split, the 3 selected classifiers were trained:

* [sklearn.tree.DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)
* [sklearn.svm.SVC](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)
* [xgboost.XGBClassifier](https://xgboost.readthedocs.io/en/latest/python/python_api.html)

*Note: the initial classification was done using each classifier’s default hyper-parameters*

With each classifier model trained, the 20% test split data was then used to evaluate and compare each model’s performance using sklearn’s Log Loss.

At this stage, the following results were obtained:

|  |  |
| --- | --- |
| Classifier | Log Loss Score |
| Decision Tree | 11.5973 |
| SVM | 1.1074 |
| XGBoost | 0.8940 |

Table 4: Log Loss scores for classifiers (pre-refinement)

One of the challenges faced at this point was the slow performance of the SVM classifier training. Even when using the more efficient label encoding approach (as discussed in the [Data Preprocessing](#_Data_Preprocessing) section), the algorithm was taking > 5 mins to train. On the other hand, both the Decision Tree and the XGBoost classifiers were able to complete their training within a few seconds. This also meant that any refinement attempted to be done on the SVM was extremely slow (as expected, due to the classifier’s level of complexity[[8]](#footnote-8). In addition, XGBoost’s score was almost 20% better than SVM’s.

In the end, XGBoost performed significantly better out-of-the-box, which made it an ideal candidate for refinement / fine-tuning[[9]](#footnote-9). This refinement process also posed a bit of a challenged, as a manual approach had to be taken. The reason was that the XGBoost library used for the analysis was not part of sklearn’s core library, so it had to be installed as a third-party library. Unfortunately, the version used was missing some API integrations with the rest of sklearn, including the ability to use sklearn’s GridSearchCV functionality. Nonetheless, a methodic approach to the refinement enabled the model to be successfully optimized.

## Refinement

For the refinement stage, the XGBoost model was selected. Also, a manual approach similar to sklearn’s [GridSearchCV](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) was followed as previously explained. The process considered the following hyper-parameters[[10]](#footnote-10):

* learning\_rate
* max\_depth
* min­\_child\_weight
* gamma
* subsample
* colsample\_bytree
* nthread

The process to fine-tune the classifier’s hyper-parameters consisted of tweaking the hyper-parameter values (one at-a-time), training a new classifier with the tweaked hyper-parameters, and testing the new Log Loss score for the updated model. The order in which these hyper-parameters were tuned was the following:[[11]](#footnote-11)

1. Tweak learning\_rate
2. Tweak nthread
3. Tweak max\_depth
4. Tweak subsample and colsample\_bytree
5. Tweak min\_child\_weight
6. Tweak gamma

In some cases (for some hyper-parameters) the new values would actually generate worse Log Loss scores. In those cases, preference was given to the corresponding default hyper-parameter value. The following table summarizes some of the intermediate hyper-parameter values and Log Loss scores obtained during the refinement[[12]](#footnote-12):

|  |  |
| --- | --- |
| Hyper-parameters | Log Loss |
| learning\_rate = 0.1  max\_depth = 5  min\_child\_weight = 1  gamma = 0  subsample = 0.8  colsample\_bytree = 0.8  nthread = 4 | 0.887322965214 |
| learning\_rate = 0.1  max\_depth = 8  min\_child\_weight = 1  gamma = 0  subsample = 0.8  colsample\_bytree = 0.8  nthread = 4 | 0.897113135346 |
| learning\_rate = 0.1  max\_depth = 5  min\_child\_weight = 10  gamma = 0  subsample = 0.8  colsample\_bytree = 0.8  nthread = 4 | 0.888368358771 |
| learning\_rate = 0.1  max\_depth = 5  min\_child\_weight = 1  gamma = 0  subsample = 1.0  colsample\_bytree = 0.8  nthread = 4 | 0.886711304789 |
| learning\_rate = 0.1  max\_depth = 5  min\_child\_weight = 1  gamma = 0  subsample = 0.95  colsample\_bytree = 0.8  nthread = 4 | 0.887158694553 |
| learning\_rate = 0.1  max\_depth = 5  min\_child\_weight = 1  gamma = 0  subsample = 0.905  colsample\_bytree = 0.8  nthread = 4 | 0.886223103423 |

Table 5: Refinement Process Samples

At the end of the refinement process, the Log Loss score attained was 0.886223103423, as compared to the initial (un-refined) value of 0.89408148122.

# Results

## Model Evaluation and Validation

As previously noted, the Decision Tree classifier performed poorly as compared with the SVM and XGBoost classifiers. The difference is significant (> 10 Log Loss points between the Decision Tree and the SVM models), which deemed that algorithm unusable for our predictions.

On the other hand, both models generated by SVM and XGBoost showed low Log Loss scores out-of-the-box. In the end, XGBoost was picked for further refinement for the reasons previously explained in the [Implementation](#_Implementation) section.

After several cycles of trial-and-error, the refinement process generated the following results:

|  |  |
| --- | --- |
| Item | Value |
| learning\_rate | 0.1 |
| max\_depth | 5 |
| min\_child\_weight | 1 |
| gamma | 0 |
| subsample | 0.905 |
| colsample\_bytree | 0.8 |
| nthread | 4 |
| Log Loss Before Refinement | 0.8941 |
| Log Loss After Refinement | 0.8862 |

Table 6: Evaluation and Validation Results

## Justification

The scores generated pre- and post-refinement by XGBoost were both better than our Benchmark. Moreover, while refining the model, several submissions were done to Kaggle’s evaluation engine. At best, the score generated by the engine classified the model within the top third of its [public board](https://www.kaggle.com/c/shelter-animal-outcomes/leaderboard) (around the 500 position). Given the data used to make these submissions were entirely new and not at all seen by the training stage, this proved that our model has the capacity to generalize predictions to classify unseen data.

# Conclusion

## Free-form Visualization

It’s interesting to visualize the difference between the OutcomeType counts in the dataset. This shows that most of the animals are (fortunately) adopted. On the other hand, “Died” and “Euthanasia” are the lowest, but the latter still has a representative level, and this presumably is a good reason for the shelter to try to devote more efforts to shift those numbers more towards the Adoption outcome.

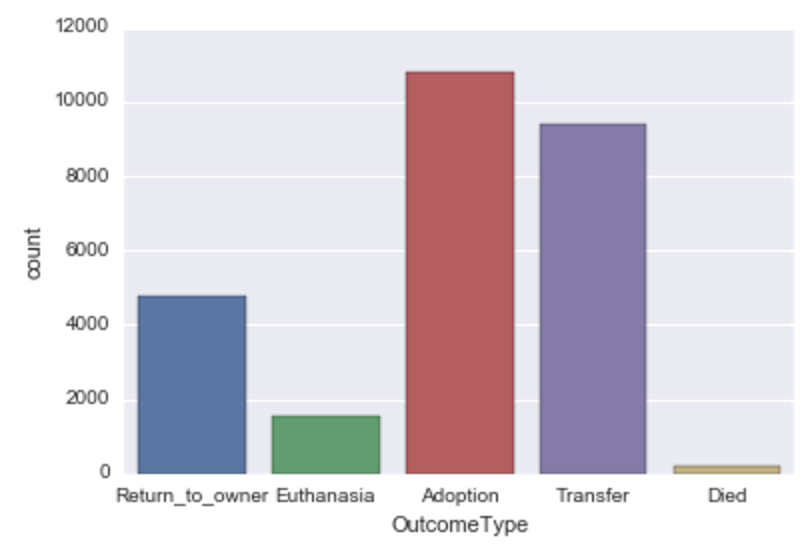


Figure 5: OutcomeType count

The prediction model obtained in this analysis has the capability to help the shelter devote efforts to improve those outcomes.

## Reflection

The goal of this project was to predict the outcome for animals in the shelter, based on the type of animal, sex, age, breed and color, with an accuracy of < 1.0 Log Loss.

One of the challenges faced was the fact that the values of most features were categories, and hence non-quantifiable by nature. Moreover, the uniqueness of these values was very large, making the number of permutations in the training data big. Fortunately, the train data we had available for the project was also large (>26K records), and this helped better train our models. Also, by utilizing the label encoding method, it was possible to keep the number of feature columns under control after normalizing the data.

Another challenged encountered was the slow performance of the SVM algorithm for a large and varied dataset such as the one we had. Training and refining our SVM model proved to be extremely slow and impractical. Fortunately, XGBoost was a much better alternative.

From the algorithms we tried for our problem space, XGBoost was by far the best in terms of accuracy and speed of training/testing.

At the end of this analysis, we were able to produce a classification model with a low Log Loss score that could help the shelter make future predictions for their animals.

## Improvement

There are multiple ways we could improve the accuracy of the predictions. The most obvious one is to try using other classifiers. One that comes to mind is Neural Networks, more specifically those involving deep-learning techniques, given the high potential they’ve shown on all kinds of problem spaces in recent years. It would be worth trying to use a multi-layer neural network to create a prediction model for this scenario.

Another improvement would be to further pre-process the train dataset to make slightly broader categories for certain features. For example, on the AgeuponOutcome features, we could create subgroups based on monthly ages, instead of daily ages, and tag each sample with the corresponding age subgroup. Similarly, subgroups of Breeds could be established among the population. For example, in the case of dogs, all breeds with a Yorkshire component in them could be qualified as a single group/value; the same would be done with other “similar” breeds.

A third suggestion to improve the predictions is to divide the train dataset into dogs and cats and treat them completely separate, so that independent ML models are generated for each animal group. Obviously, this would also require the testing and prediction phases to be divided between dogs and cats, so that the appropriate model can be used for the particular case. Presumably, having “specialized” prediction models for each animal group could help improve the predictions, as the models would potentially have the opportunity to learn the nuances of each animal group without getting disturbed by bias from the other group.

Finally, as a long-term strategy for the shelter, it would be interesting to start tracking entirely different aspects that could eventually become features of a prediction model. For example, aspects related with the animal’s personality could potentially have a strong correlation with their outcome; things like “friendly personality”, “likes children”, “obedient”, etc. could be personality traits that may play a role in the chances of an animal getting adopted, and hence avoiding death or euthanasia.

1. Source: <https://www.kaggle.com/c/shelter-animal-outcomes> [↑](#footnote-ref-1)
2. Source: <https://www.kaggle.com/wiki/LogarithmicLoss> [↑](#footnote-ref-2)
3. See the [Improvement](#_Improvement) section in the Conclusion for more suggestions on improving the prediction model. [↑](#footnote-ref-3)
4. In simple terms, if one can think of a half-pepperoni-half-veggies pizza representing the outcome space of a 2-class problem, the hyperplane dividing the outcome classes would be represented by a straight line crossing the middle of the pizza. This example obviously assumes the classes are “linearly” separable. [↑](#footnote-ref-4)
5. The Kernel trick can be thought of as a mathematical mechanism to define decision boundaries for non-linearly separable outcome classes. It works by transforming the data in such a way that it is represented spatially differently (yet equivalently), so that a linear hyperplane can then divide its outcome classes. [↑](#footnote-ref-5)
6. <https://en.wikipedia.org/wiki/Gradient_boosting> [↑](#footnote-ref-6)
7. An interesting post to this matter is available in the competition’s forum: <https://www.kaggle.com/c/shelter-animal-outcomes/forums/t/22119/cheating-your-way-to-the-top-of-the-lb-remove-the-lb> [↑](#footnote-ref-7)
8. Several attempts to perform GridSearchCV analysis on the SVM model were made, but all were stopped after having the process run several hours without any definitive results. In one particular case, the process was left running for 24 hours before stopping it. [↑](#footnote-ref-8)
9. The results of the comparison are detailed in the [Model Evaluation and Validation](#_Model_Evaluation_and) section of this document. [↑](#footnote-ref-9)
10. Official documentation on XGBoost parameters and their meaning: <https://github.com/dmlc/xgboost/blob/master/doc/parameter.md> [↑](#footnote-ref-10)
11. Inspired by: <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/> [↑](#footnote-ref-11)
12. Most intermediate values have not been included for brevity. [↑](#footnote-ref-12)