Machine Learning Engineer Nanodegree

Capstone Project

Rafael Torres

January 15, 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]. 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 featured in Kaggle.

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 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 define a classification model based on data collected by the shelter, which could help them better predict the outcome of their animals. Such information can in turn help them identify segments of their animal population that need extra help in finding new homes.

Metrics

Kaggle's evaluation engine 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.

[1] Source: https://www.kaggle.com/c/shelter-animal-outcomes

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

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

Equation 1: Log Loss

Where:

- *N = number of examples*
- *M = number of classes*
- *y_{ij} = 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` via the `sklearn.metrics.log_loss` 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.

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*

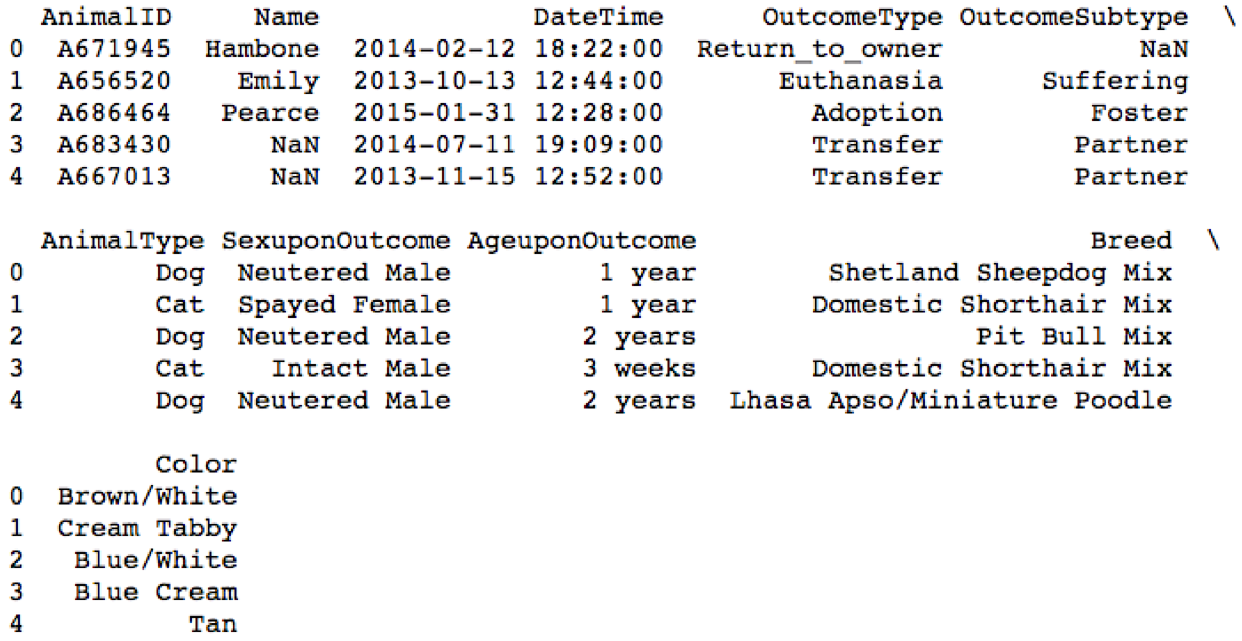
[2] Source: https://www.kaggle.com/wiki/LogarithmicLoss

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

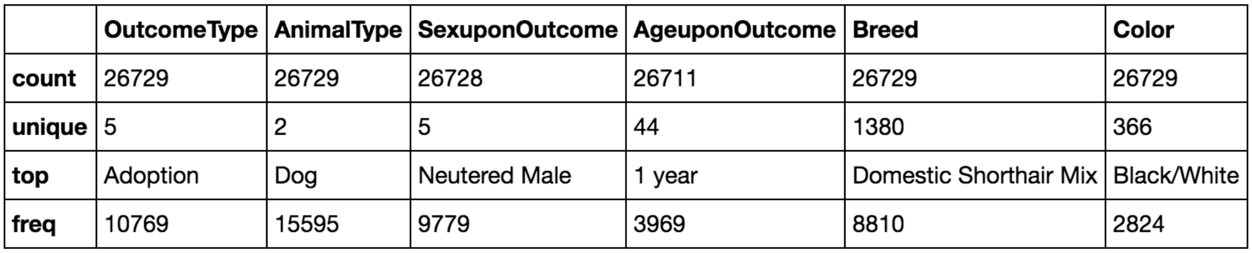
Here’s a summary of the shape of the dataset:



```
  AnimalID     Name             DateTime      OutcomeType OutcomeSubtype  \
0  A671945  Hambone  2014-02-12 18:22:00  Return_to_owner            NaN
1  A656520    Emily  2013-10-13 12:44:00       Euthanasia      Suffering
2  A686464   Pearce  2015-01-31 12:28:00         Adoption         Foster
3  A683430      NaN  2014-07-11 19:09:00         Transfer        Partner
4  A667013      NaN  2013-11-15 12:52:00         Transfer        Partner

  AnimalType SexuponOutcome AgeuponOutcome                        Breed  \
0        Dog  Neutered Male         1 year        Shetland Sheepdog Mix
1        Cat  Spayed Female         1 year       Domestic Shorthair Mix
2        Dog  Neutered Male        2 years                 Pit Bull Mix
3        Cat    Intact Male        3 weeks       Domestic Shorthair Mix
4        Dog  Neutered Male        2 years  Lhasa Apso/Miniature Poodle

         Color
0  Brown/White
1  Cream Tabby
2   Blue/White
3   Blue Cream
4          Tan
```

Table 1: Panda's `DataFrame` head of train dataset

And here’s a broad description of the dataset:



	OutcomeType	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
count	26729	26729	26728	26711	26729	26729
unique	5	2	5	44	1380	366
top	Adoption	Dog	Neutered Male	1 year	Domestic Shorthair Mix	Black/White
freq	10769	15595	9779	3969	8810	2824

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:

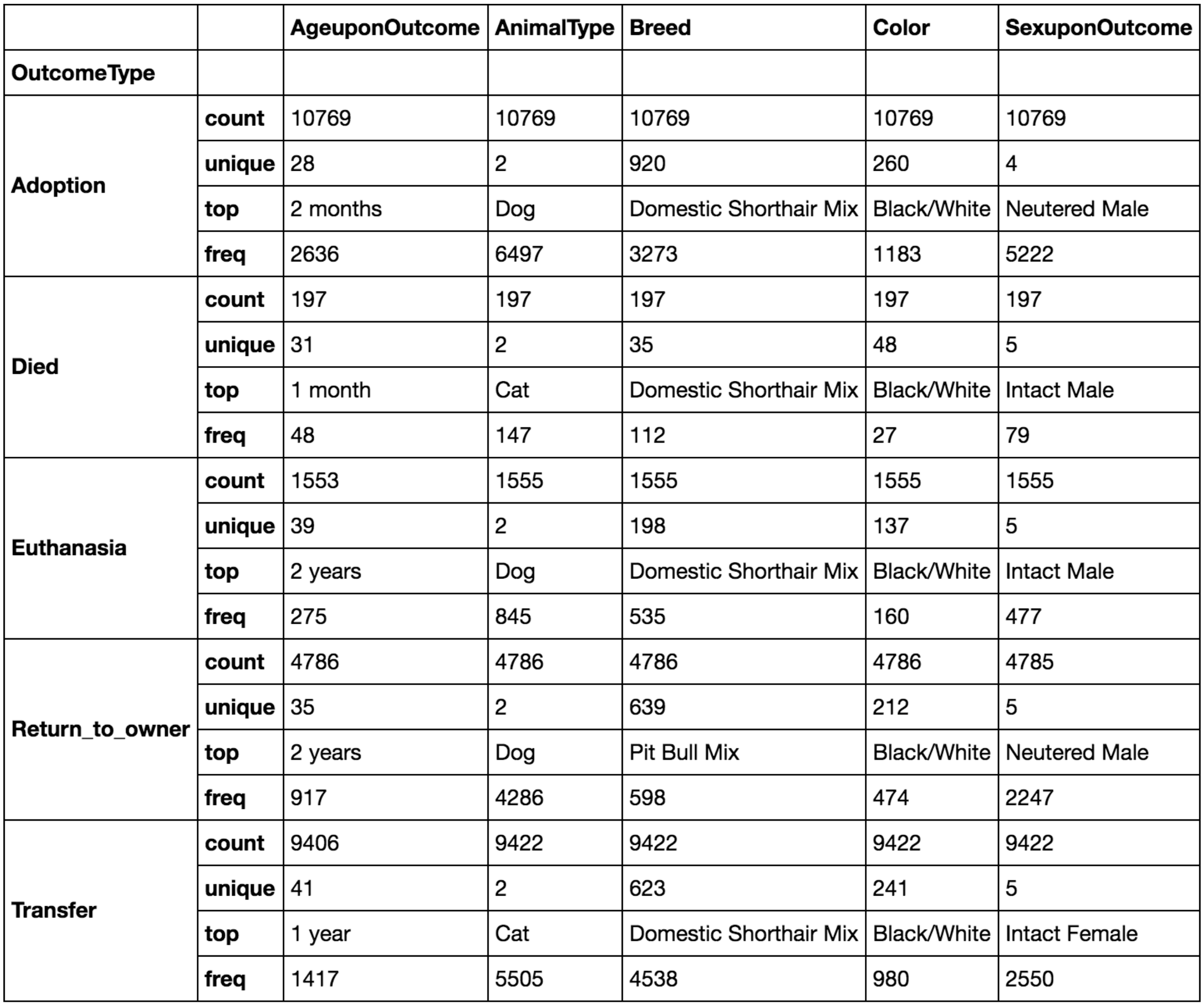
- Features: `AnimalType`, `SexuponOutcome`, `AgeuponOutcome`, `Breed`, `Color`.
- Target: `OutcomeType`.

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:



		AgeuponOutcome	AnimalType	Breed	Color	SexuponOutcome
OutcomeType						
Adoption	**count**	10769	10769	10769	10769	10769
	unique	28	2	920	260	4
	top	2 months	Dog	Domestic Shorthair Mix	Black/White	Neutered Male
	freq	2636	6497	3273	1183	5222
Died	**count**	197	197	197	197	197
	unique	31	2	35	48	5
	top	1 month	Cat	Domestic Shorthair Mix	Black/White	Intact Male
	freq	48	147	112	27	79
Euthanasia	**count**	1553	1555	1555	1555	1555
	unique	39	2	198	137	5
	top	2 years	Dog	Domestic Shorthair Mix	Black/White	Intact Male
	freq	275	845	535	160	477
Return_to_owner	**count**	4786	4786	4786	4786	4785
	unique	35	2	639	212	5
	top	2 years	Dog	Pit Bull Mix	Black/White	Neutered Male
	freq	917	4286	598	474	2247
Transfer	**count**	9406	9422	9422	9422	9422
	unique	41	2	623	241	5
	top	1 year	Cat	Domestic Shorthair Mix	Black/White	Intact Female
	freq	1417	5505	4538	980	2550

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 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:

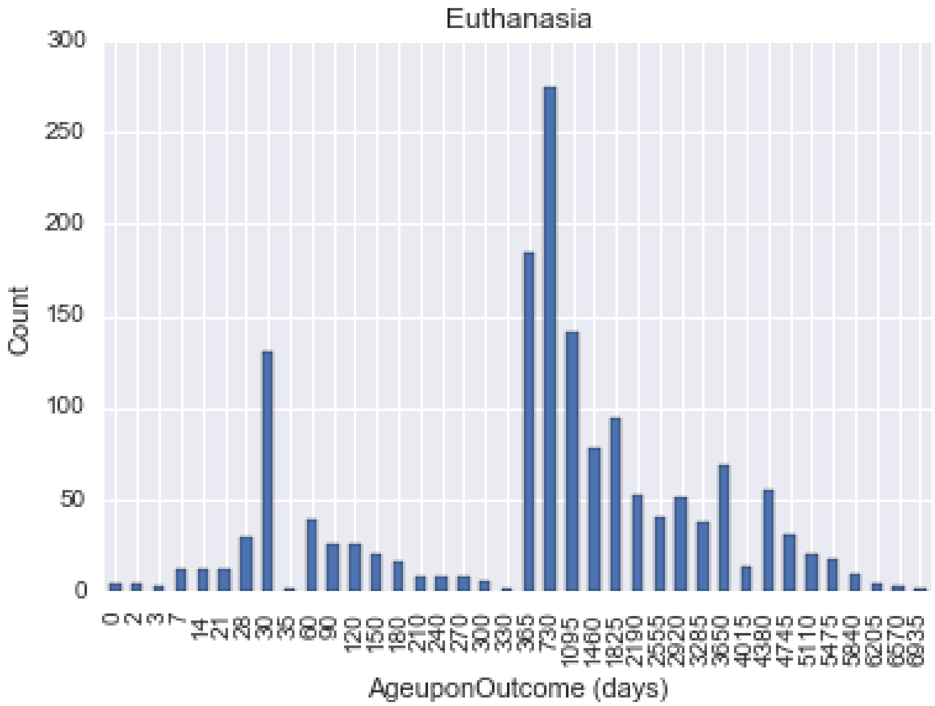


Figure 1: "Euthanasia" `OutcomeType` *count vs* `AgeuponOutcome` *Bar Graph*

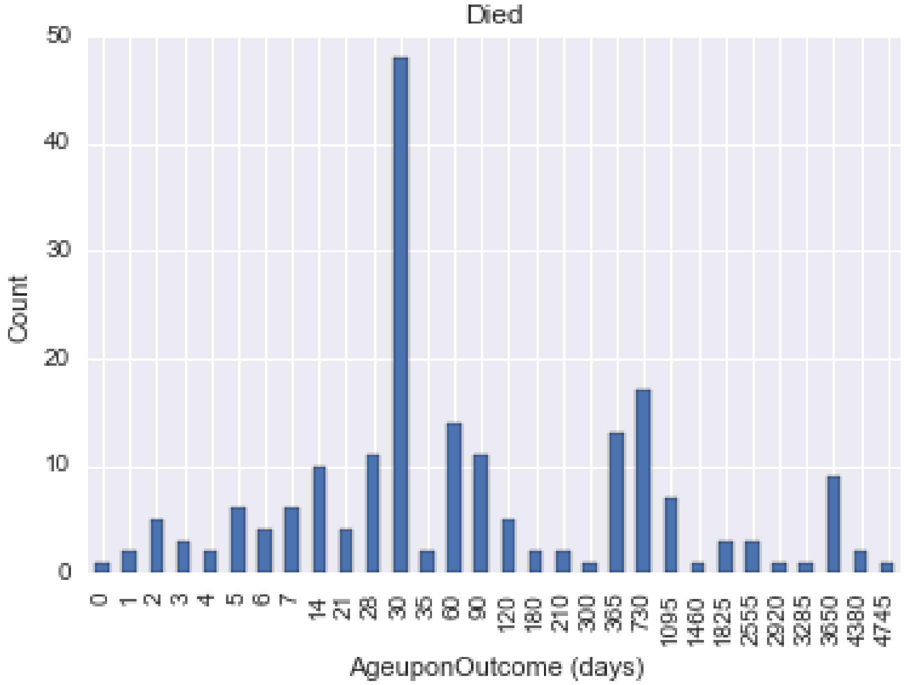


Figure 2: "Died" `OutcomeType` *count vs* `AgeuponOutcome` *Bar Graph*

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].

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.

[3] See the Improvement section in the Conclusion for more suggestions on improving the prediction model.

- 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.
- 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[4]. It turned out that at the end 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[5].

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 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 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

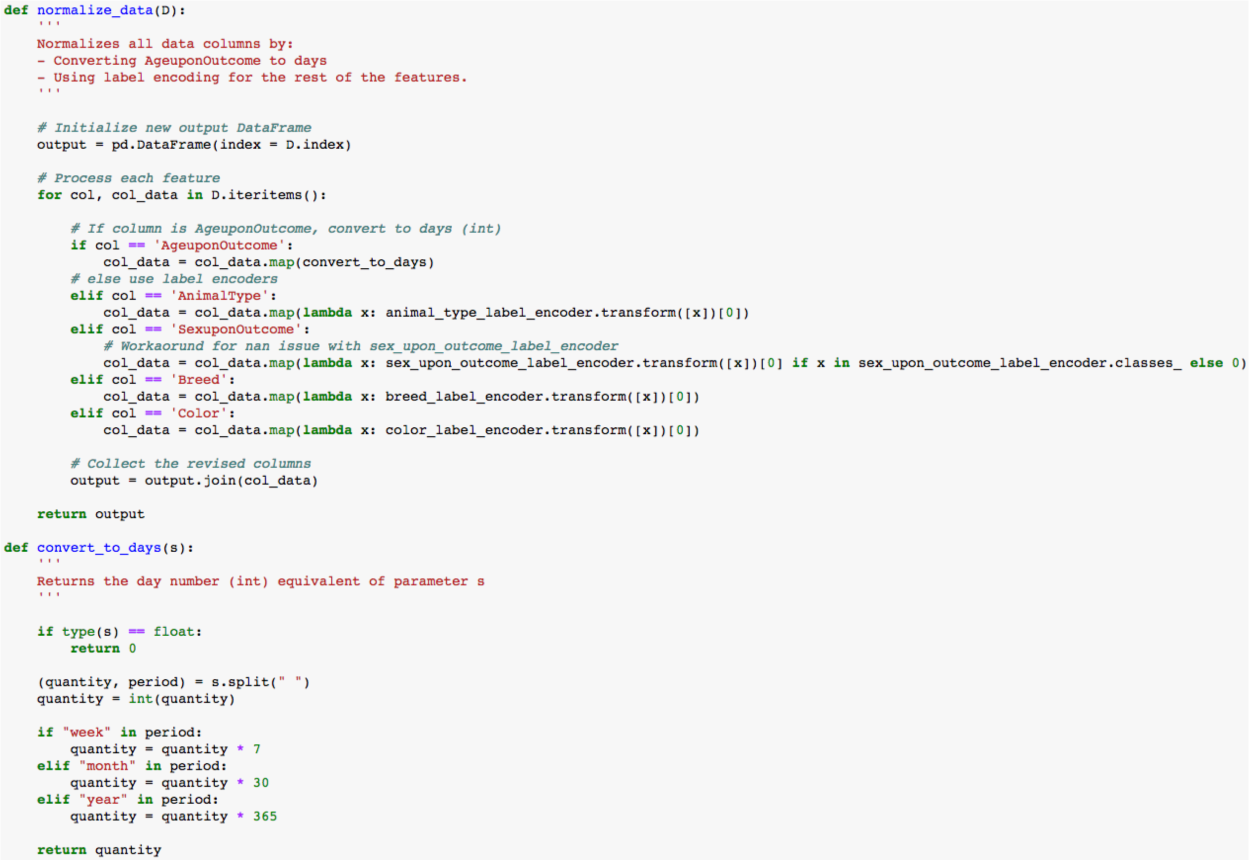
[4] https://en.wikipedia.org/wiki/Gradient_boosting

[5] 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

approach was to use sklearn's `LabelEncoder`, 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.

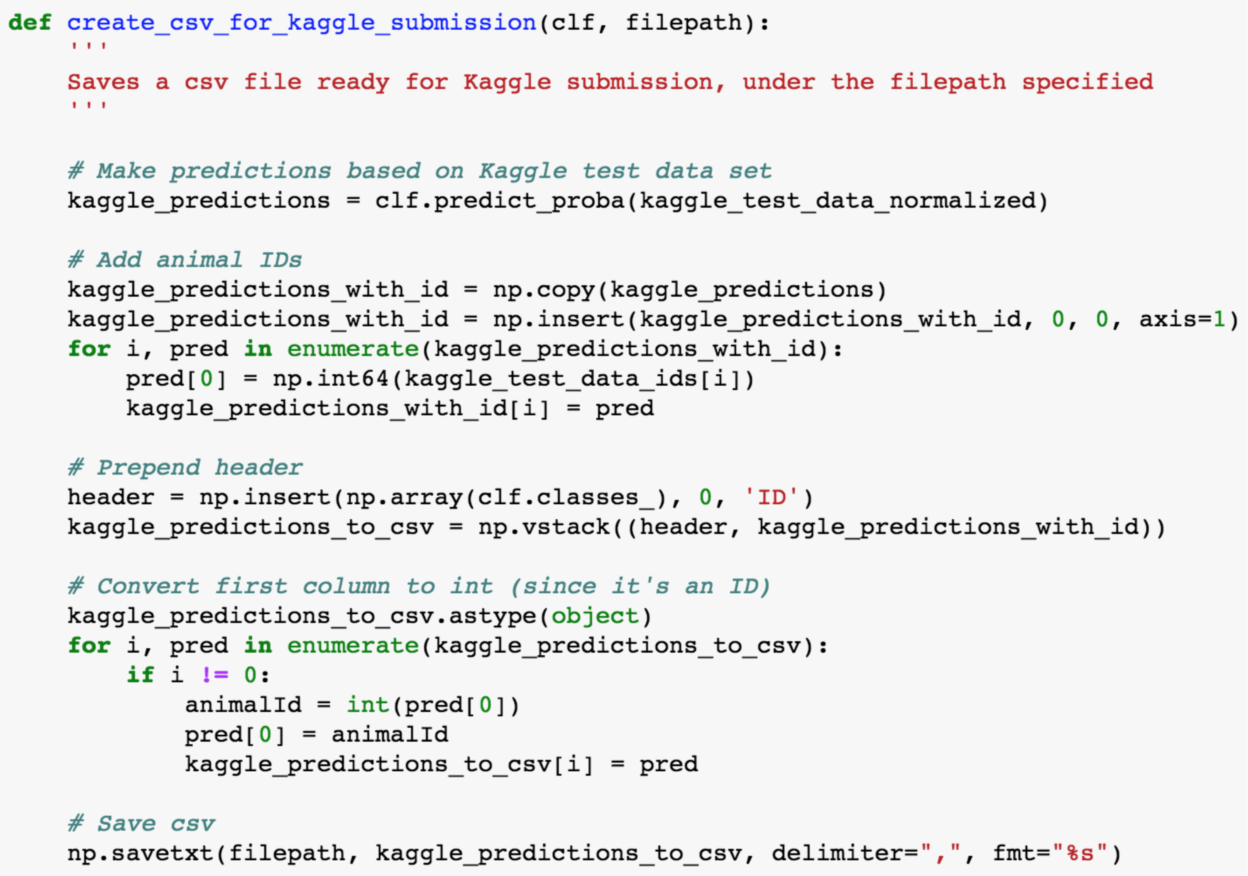


```python
def normalize_data(D):
    '''
    Normalizes all data columns by:
    - Converting AgeuponOutcome to days
    - Using label encoding for the rest of the features.
    '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = D.index)

    # Process each feature
    for col, col_data in D.iteritems():

        # If column is AgeuponOutcome, convert to days (int)
        if col == 'AgeuponOutcome':
            col_data = col_data.map(convert_to_days)
        # else use label encoders
        elif col == 'AnimalType':
            col_data = col_data.map(lambda x: animal_type_label_encoder.transform([x])[0])
        elif col == 'SexuponOutcome':
            # Workaorund for nan issue with sex_upon_outcome_label_encoder
            col_data = col_data.map(lambda x: sex_upon_outcome_label_encoder.transform([x])[0] if x in sex_upon_outcome_label_encoder.classes_ else 0)
        elif col == 'Breed':
            col_data = col_data.map(lambda x: breed_label_encoder.transform([x])[0])
        elif col == 'Color':
            col_data = col_data.map(lambda x: color_label_encoder.transform([x])[0])

        # Collect the revised columns
        output = output.join(col_data)

    return output

def convert_to_days(s):
    '''
    Returns the day number (int) equivalent of parameter s
    '''

    if type(s) == float:
        return 0

    (quantity, period) = s.split(" ")
    quantity = int(quantity)

    if "week" in period:
        quantity = quantity * 7
    elif "month" in period:
        quantity = quantity * 30
    elif "year" in period:
        quantity = quantity * 365

    return quantity
```

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 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` to manipulate data arrays and export them to a `csv` file, as shown below:



```
def create_csv_for_kaggle_submission(clf, filepath):
    '''
    Saves a csv file ready for Kaggle submission, under the filepath specified
    '''

    # Make predictions based on Kaggle test data set
    kaggle_predictions = clf.predict_proba(kaggle_test_data_normalized)

    # Add animal IDs
    kaggle_predictions_with_id = np.copy(kaggle_predictions)
    kaggle_predictions_with_id = np.insert(kaggle_predictions_with_id, 0, 0, axis=1)
    for i, pred in enumerate(kaggle_predictions_with_id):
        pred[0] = np.int64(kaggle_test_data_ids[i])
        kaggle_predictions_with_id[i] = pred

    # Prepend header
    header = np.insert(np.array(clf.classes_), 0, 'ID')
    kaggle_predictions_to_csv = np.vstack((header, kaggle_predictions_with_id))

    # Convert first column to int (since it's an ID)
    kaggle_predictions_to_csv.astype(object)
    for i, pred in enumerate(kaggle_predictions_to_csv):
        if i != 0:
            animalId = int(pred[0])
            pred[0] = animalId
            kaggle_predictions_to_csv[i] = pred

    # Save csv
    np.savetxt(filepath, kaggle_predictions_to_csv, delimiter=",", fmt="%s")
```

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

Implementation

To train the models, the dataset in *train.csv* (after being pre-processed via the method described above) was split in 80/20 (train/test data).

Using the 80% train split, the 3 selected classifiers were trained with their corresponding default hyper-parameters:

- sklearn.tree.DecisionTreeClassifier

- sklearn.svm.SVC
- xgboost.XGBClassifier

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.

In the end, XGBoost performed significantly better out-of-the-box, which made it an ideal candidate for refinement / fine-tuning[6].

Refinement

For the refinement stage, the XGBoost model was selected. Also, a manual approach similar to sklearn's `GridSearchCV` was followed[7]. The process considered the following hyper-parameters[8]:

- `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:[9]

a) Tweak `learning_rate`
b) Tweak `nthread`
c) Tweak `max_depth`
d) Tweak `subsample` and `colsample_bytree`
e) Tweak `min_child_weight`
f) Tweak `gamma`

[6] The results of the comparison are detailed in the Model Evaluation and Validation section of this document.

[7] The XGBoost library used for this analysis is not part of sklearn's core library, so it had to be installed as a plugin. Unfortunately, the version used is missing some API integrations with the rest of `sklearn`, including the ability to use sklearn's `GridSearchCV` functionality.

[8] Official documentation on XGBoost parameters and their meaning: https://github.com/dmlc/xgboost/blob/master/doc/parameter.md

[9] Inspired by: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

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.

Results

Model Evaluation and Validation

With the trained models, each model's Log Loss was measured using the 20% test split data. The following were the scores obtained at this stage:

Classifier	Log Loss Score
Decision Tree	11.5973
SVM	1.1074
XGBoost	0.8940

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

As it can be observed, the Decision Tree classifier performed poorly as compared with the other 2. The difference is significant, which deemed that model/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. One big disadvantage between the 2 was that the SVM training process very slow (as expected based on the classifier's complexity). This hinted that any kind of refinement via `GridSearchCV` of that model would turn out to be extremely slow[10]. In addition, XGBoost's score was almost 20% better than SVM's. The combination of both factors led to the conclusion that the only model worthy of any refinement attempt was the one produced by XGBoost.

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

Item	Value
learning_rate	0.1
max_depth	5

[10] 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.

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 5: Post-refinement 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 (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.

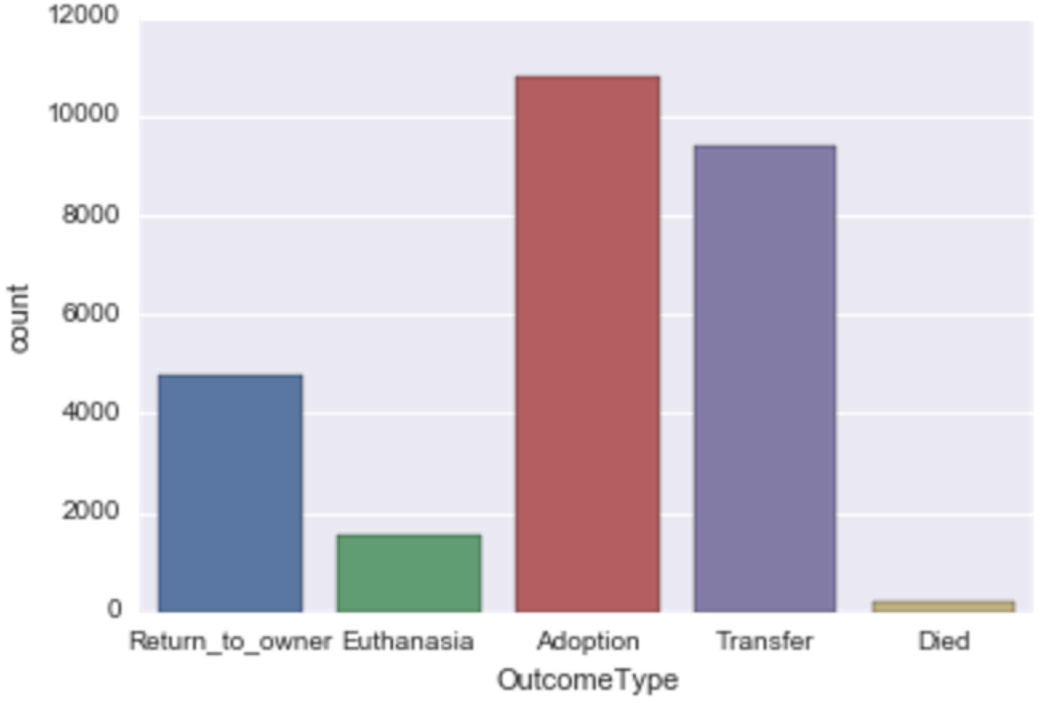


Figure 5: `OutcomeType` *count*

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.