Machine Learning Engineer Nanodegree

Capstone Project

Rahul Trada
January 10, 2019

I. Definition

Project Overview

Every year, approximately 7.6 million companion animals end up in US shelters. Many animals are given up as unwanted by their owners, while others are picked up after getting lost or taken out of cruelty situations. Many of these animals find forever families to take them home, but just as many are not so lucky. 2.7 million dogs and cats are euthanized in the US every year.

With machine learning we could analyze data relating to the outcome of animals based on features of these animals. These insights could help shelters focus their energy on specific animals that need a little extra help in finding a new home.

Problem Statement

In this project, we look at a Kaggle dataset provided by the Austin Animal Center, where data is given on intake information of the animals including breed, color, sex and age, and we will predict the outcomes of these animals. The outcomes represent the status of animals as they leave the Animal Center, and include: Adoption, Died, Euthanasia, Return to Owner, and Transfer. I will use machine learning methods learned in the nanodegree program to perform this prediction task.

Metrics

Since this is a multi-class classification problem, the metric we will use is the multi-class log loss. For each animal, I will submit a set of predicted probabilities (one for each class). The reason the multi-class log loss is appropriate is because it will penalize not just the wrong class prediction, but also the correct class prediction when it has low probability. Therefore we

need to predict the correct class with high probability to get a good score. The formula is:

$$logloss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=0}^{M} y_{ij}\log(p_{ij})$$

where N is the number of animals in the test set, M is the number of outcomes, *log* is the natural logarithm, y_{ij} is 1 if observation i is in outcome j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to outcome j.

II. Analysis

Data Exploration

To begin with data exploration, we first load the data into a pandas DataFrame, and look at the first five rows of the data.

```
In [38]: data.head()
```

Out[38]:

	AnimalID	Name	DateTime	OutcomeType	OutcomeSubtype	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
0	A671945	Hambone	2014-02-12 18:22:00	Return_to_owner	NaN	Dog	Neutered Male	1 year	Shetland Sheepdog Mix	Brown/White
1	A656520	Emily	2013-10-13 12:44:00	Euthanasia	Suffering	Cat	Spayed Female	1 year	Domestic Shorthair Mix	Cream Tabby
2	A686464	Pearce	2015-01-31 12:28:00	Adoption	Foster	Dog	Neutered Male	2 years	Pit Bull Mix	Blue/White
3	A683430	NaN	2014-07-11 19:09:00	Transfer	Partner	Cat	Intact Male	3 weeks	Domestic Shorthair Mix	Blue Cream
4	A667013	NaN	2013-11-15 12:52:00	Transfer	Partner	Dog	Neutered Male	2 years	Lhasa Apso/Miniature Poodle	Tan

- We have 26729 records, with 10 columns.
- **AnimalID** is a unique identifier, not needed for our purposes; I'll drop this column.
- **Name** is the name of the animal, but we already see some null values
- **DateTime** corresponds to when the Outcome occurred. This is an issue because we cannot know the time of the outcome before making the prediction. There are various discussions on Kaggle about this being a data leak, and that the best scoring models on Kaggle all took advantage of using this column, which is cheating. Therefore I'll drop this column.
- **OutcomeType** is the target variable. This is what we are trying to predict.
- **OutcomeSubtype** is an elaboration of the outcome. Since we are only predicting outcome, this is not useful for us, so I'll drop it.

- **AnimalType** is one of 2 values: Cat or Dog.
- **SexuponOutcome** identifies whether the animal is male or female, as well as whether it is neutered/spayed or intact.
- **AgeuponOutcome** is the age of the animal at the time of the outcome.
- **Breed** is the animal's breed.
- **Color** is the color of its fur.

Here are some basic statistics of the remaining data.

```
data.describe()
```

	Name	OutcomeType	AnimalType	SexuponOutcome	AgeuponOutcome	Breed	Color
count	19038	26729	26729	26728	26711	26729	26729
unique	6374	5	2	5	44	1380	366
top	Max	Adoption	Dog	Neutered Male	1 year	Domestic Shorthair Mix	Black/White
freq	136	10769	15595	9779	3969	8810	2824

All the variables are categorical. Breed and Color will be tricky as there are far too many unique values. We'll need to find a way to simplify these variables if they are to be used by our models for prediction. We can convert AgeuponOutcome to a numeric variable with some data processing.

Let's check if there are any missing values.

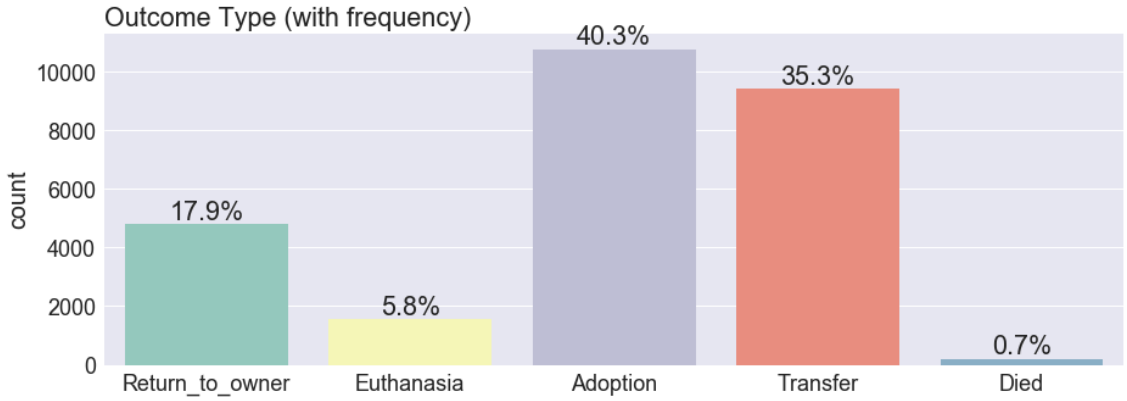
```
data.isna().sum()
```

```
Name              7691
OutcomeType          0
AnimalType           0
SexuponOutcome       1
AgeuponOutcome      18
Breed                0
Color                0
dtype: int64
```

We have a significant number of animals without a name. This variable could potentially be important because it could indicate a stray animal as opposed to one that belonged to a home, which could influence future adoption etc. We could create a binary variable 'HasName' for use in the model from this, later in data processing. For SexuponOutcome and AgeuponOutcome, there are very few records with missing values, so I'll fill them in with the most common values (the mode of the column).

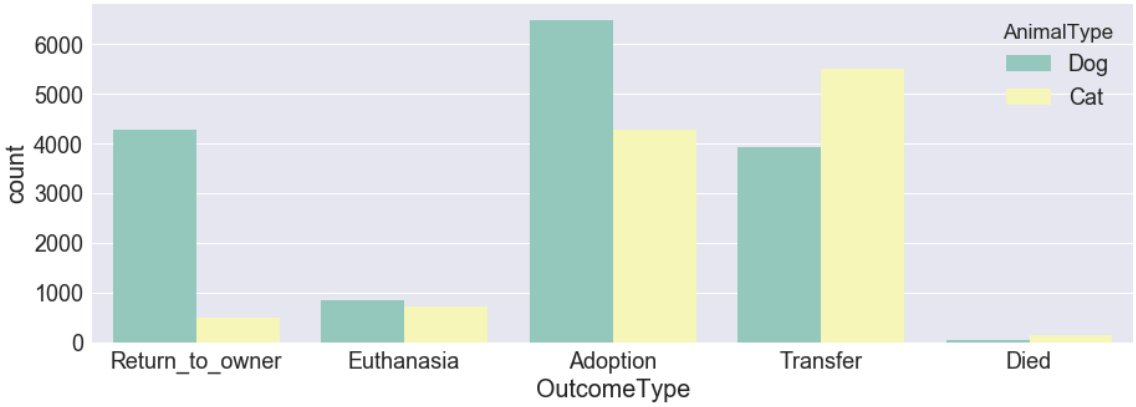
Exploratory Visualization

First let's visualize the different outcomes of the animals in the data



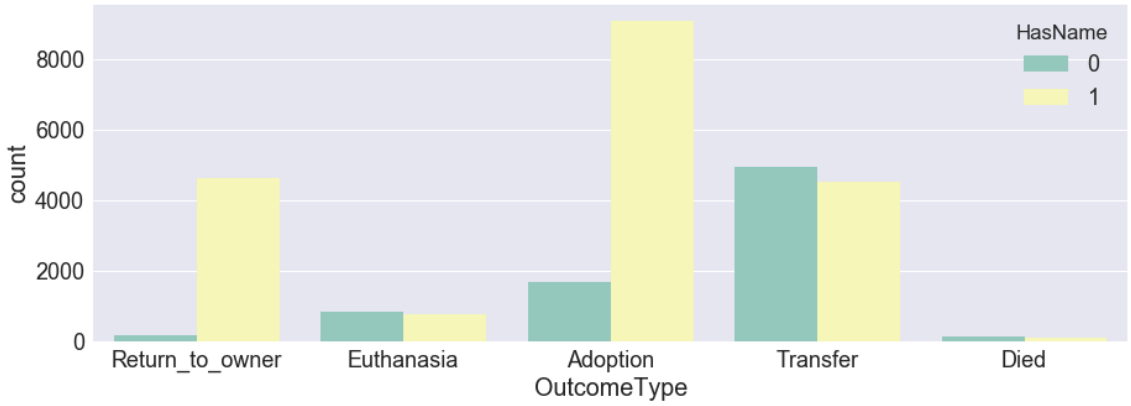
So the good news is the 75.6% of the animals are either adopted, or transferred to another place, and only very few animals (<1%) die while at the shelter, though a minority (5.8%) do get euthanized.

Let's look at the outcomes based on a particular feature, say AnimalType.

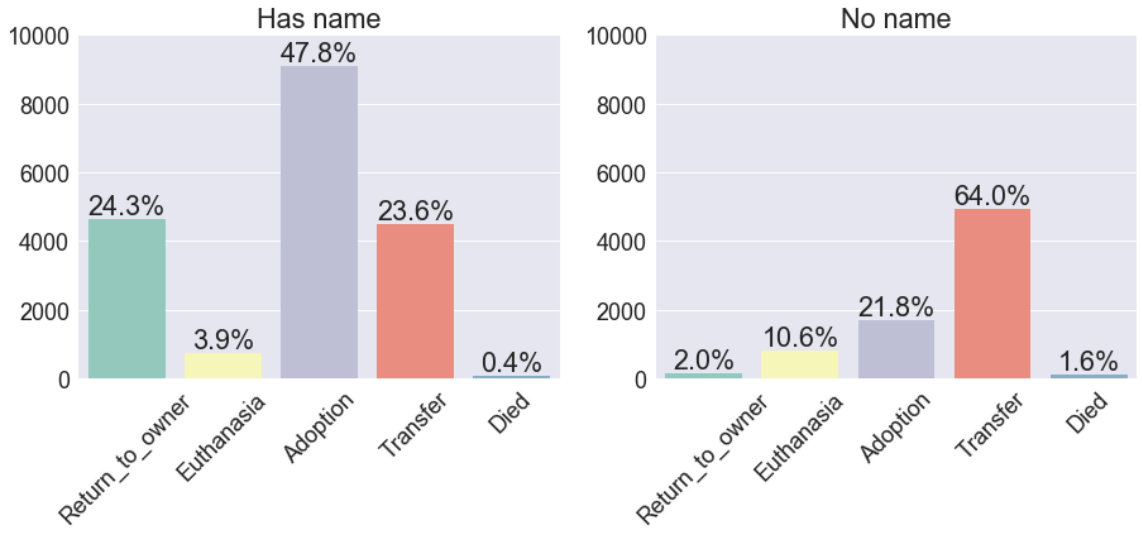


Here we see that more dogs are adopted compared to cats, but more cats are transferred. Interestingly, many more dogs are returned to owner compared to cats. The AnimalType variable looks like it could be very important in predicting the outcome.

Let's now take a look at the Name variable. There are too many unique names in the data, but a significant number of animals don't have a name. So to keep things simple, I want to compare the outcomes for named vs non-named animals. To do this, I wrote a function that will change the Name column into a binary column HasName, with a 1 if the animal has a name, and 0 otherwise.

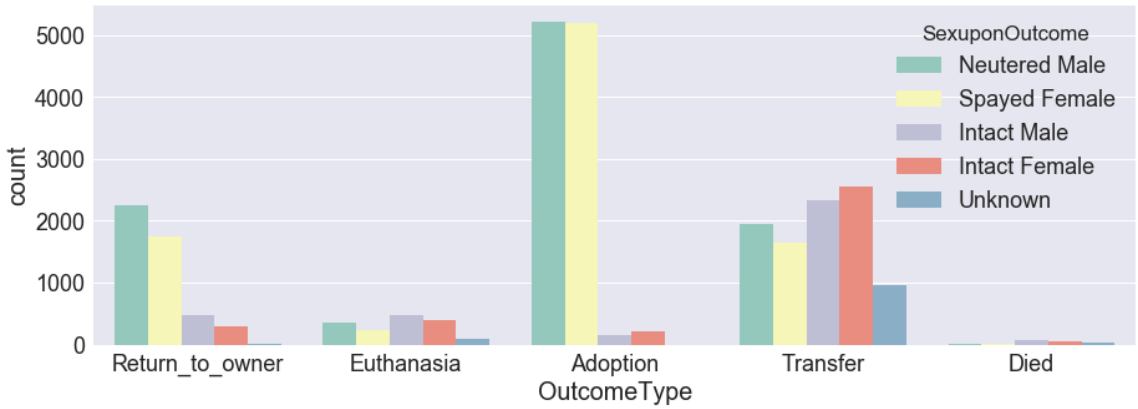


So we see that more named animals are returned to owner and adopted. But there were more named vs unnamed animals to begin with, and I want to make sure the difference we're seeing is not simply due to that fact. So I want to see the proportion of each group for each outcome. So let's create 2 more plots with the frequencies for each group included, 1 for named animals, and 1 for unnamed animals, so we can compare the outcomes between plots.



Now we can see clearly that if the animal has a name, almost half of these animals are adopted, while if it doesn't, less than a quarter are adopted. Similarly, 24.3% of named animals are returned to owner, compared to just 2% for unnamed animals. This variable is potentially very important in predicting the outcome.

Now let’s look at the variable SexuponOutcome.



There are 9780 neutered males and 8820 spayed females, and more than 5000 from each category end up being adopted, with the majority of the remainder being either returned to owner, or transferred.

On the other hand, there are 3525 intact males and 3511 intact females, and almost all of them are transferred, with the second most common outcome being euthanasia. I think we can safely say that neutered/spayed animals are more likely to get a better outcome. This is useful info the shelter could use.

Algorithms and Techniques

Since this is a multiclass classification problem, we are using a log loss metric, so it matters not just that we predict the accurate class, but that we do so with high probability, in order to get the best score. Since we have mostly categorical variables, and we don't have too much data (~26k rows, with 6 features), this would be ideal for ensemble tree-based classifiers. I will be running RandomForest and XGBoost classifiers.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

Benchmark

To set a benchmark, we'll use a dummy classifier from the sklearn library. This is just a classifier that uses simple rules to make its prediction. Later we'll use real models to see if we can improve on the loss metric. Note that we are using a stratified KFold split in order to train and test the model, because of the imbalance in the 5 different outcomes we are trying to predict in the data. This should give better results than a fixed train/test split.

```
In [108]: skf = StratifiedKFold(n_splits=5, shuffle=True)
          dummy = DummyClassifier(strategy='prior')
          loss=[]
          for train_index, test_index in skf.split(X,y):
              X_train, y_train = X.loc[train_index], y.loc[train_index]
              X_test, y_test = X.loc[test_index], y.loc[test_index]
              dummy.fit(X_train, y_train)
              y_pred = dummy.predict_proba(X_test)
              loss.append(log_loss(y_test, y_pred))
          print(loss)
          print('Mean log loss for dummy model: ', np.mean(loss))

          [1.2439119676165213, 1.243822919629386, 1.2431743174628553, 1.2431743174628553, 1.243236824648198]
          Mean log loss for dummy model:  1.2434640693639634
```

The dummy classifier using the default strategy gets a log loss of ~20. But after switching to use the 'prior' strategy, it obtains its best score of 1.24. The Kaggle leaderboard has its top score at ~0.6 for those without the data exploit mentioned at the beginning (models using the data exploit can obtain a perfect 0 loss, which is unrealistic), so to jump from 20 to 1.24 is good. Let's see if we can improve from here with real models.

III. Methodology

Data Preprocessing

There were several preprocessing steps taken to prepare the data for visualization, and then to be used by our models. These include:

1) Deleting unused columns. I deleted columns ‘AnimalID’, ‘DateTime’ and ‘OutcomeSubtype’ because they are not useful for our predictin task. I also deleted column ‘Color’ because I was unable to reduce the number of unique values to a reasonable value to be useful for our model.

2) Filling null values with the mode of the column.

3) Changing a string column Name to a binary integer column HasName. Animals with a name have a value of 1, while animals without a name have a value of 0.

4) Changing a string Age column to an integer column. There were originally different units in the column, such as ‘day’, ‘week’, ‘month’, ‘year’. I converted all values to integer days.

5) Changing the Breed column to a binary IsMixBreed column. There were more than 1000 unique values in the breed column, and there wasn’t a simple way to reduce this number. I noticed that many of the animals were mixed breed, so I converted the column into a binary, 1 if the animal is mixed breed, 0 if not.

6) One-hot encoding the categorical features for use by the model.

Implementation

As explained in the benchmark section, I used a StratifiedKFold with 5 splits for training and testing. I then take a mean of the log loss on the 5 test sets as the final loss value.

The final features used were [‘Name’, ‘AnimalType’, ‘SexuponOutcome’, ‘AgeuponOutcome’, ‘IsMixBreed’]. Remember the benchmark score(loss) was 1.24, which we are trying to see if the real models can improve upon, indicating they have learned something from the data.

I started with a default RandomForestClassifier model.

```
model = RandomForestClassifier()
loss=[]
for train_index, test_index in skf.split(X,y):
    X_train, y_train = X.loc[train_index], y.loc[train_index]
    X_test, y_test = X.loc[test_index], y.loc[test_index]
    model.fit(X_train, y_train)
    y_pred = model.predict_proba(X_test)
    loss.append(log_loss(y_test, y_pred))
print(loss)
print('Mean log loss for model: ', np.mean(loss))
```

```
[1.2557486519884427, 1.3507544390191284, 1.3667122677731518, 1.3132522918599783, 1.2935886501829272]
Mean log loss for model:  1.3160112601647256
```

We get a loss of 1.32, which is actually worse than the benchmark score. We’ll need to try parameter tuning to see if we can improve its score.

Refinement

I tried a RandomForestClassifier again, but this time using GridSearchCV to test out several different parameter settings to obtain the best settings for our problem.

```
model = RandomForestClassifier()
params = {'n_estimators': [120, 500, 800],
          'criterion' : ['gini', 'entropy'],
          'min_samples_split' : [1.0, 5, 15, 100],
         }
clf = GridSearchCV(model, params, scoring = 'neg_log_loss', cv=5, verbose=2)
clf.fit(X, y)
```

```
loss=[]
for train_index, test_index in skf.split(X,y):
    X_train, y_train = X.loc[train_index], y.loc[train_index]
    X_test, y_test = X.loc[test_index], y.loc[test_index]
    y_pred = clf.predict_proba(X_test)
    loss.append(log_loss(y_test, y_pred))
    print('train loss: ', clf.score(X_train, y_train), 'test loss: ', clf.score(X_test, y_test))
print(loss)
print('Mean log loss for model: ', np.mean(loss))
```

```
train loss:  -0.8186656660401392 test loss:  -0.8330948397224891
train loss:  -0.8202595011756352 test loss:  -0.8267239862170669
train loss:  -0.8250104667669473 test loss:  -0.8077189872957321
train loss:  -0.8208010733978136 test loss:  -0.8245597109263318
train loss:  -0.8230261397772018 test loss:  -0.8156564014417345
[0.8330948397224891, 0.8267239862170669, 0.8077189872957321, 0.8245597109263318, 0.8156564014417345]
Mean log loss for model:  0.8215507851206709
```

We get a much better score of 0.82 after parameter tuning using grid search. This is quite a bit better than the benchmark.

Let's also try a different model, this time the XGBoost classifier.

```
model = XGBClassifier(objective='multi:softprob', n_jobs=-1)
loss=[]
for train_index, test_index in skf.split(X,y):
    X_train, y_train = X.loc[train_index], y.loc[train_index]
    X_test, y_test = X.loc[test_index], y.loc[test_index]
    model.fit(X_train, y_train, eval_metric='mlogloss')
    y_pred = model.predict_proba(X_test)
    loss.append(log_loss(y_test, y_pred))
print(loss)
print('Mean log loss for model: ', np.mean(loss))
```

```
[0.8429836325410812, 0.854020893450444, 0.8686741387717237, 0.8616905970849146, 0.8517967569537205]
Mean log loss for model:  0.8558332037603769
```

With default values we get a log loss of 0.86, which is not too bad, but worse than the tuned RandomForest. I tried running the grid search on this XGBoost model as well to tune it, but it was so much slower than the RandomForest that I eventually had to abort the kernel for time constraints' sake. (We could be limited by my computer's memory and cpu limitations as well, I'm not completely sure). Regardless, I have settled on the RandomForest as the best model.

IV. Results

Model Evaluation and Validation

Benchmark

- log loss score of 1.24

XGBoost

- log loss score of 0.85

Tuned RandomForest model

- log loss score of 0.82. Training and testing scores are also virtually identical at 0.82 indicating the model has generalized well and is not overfitting.

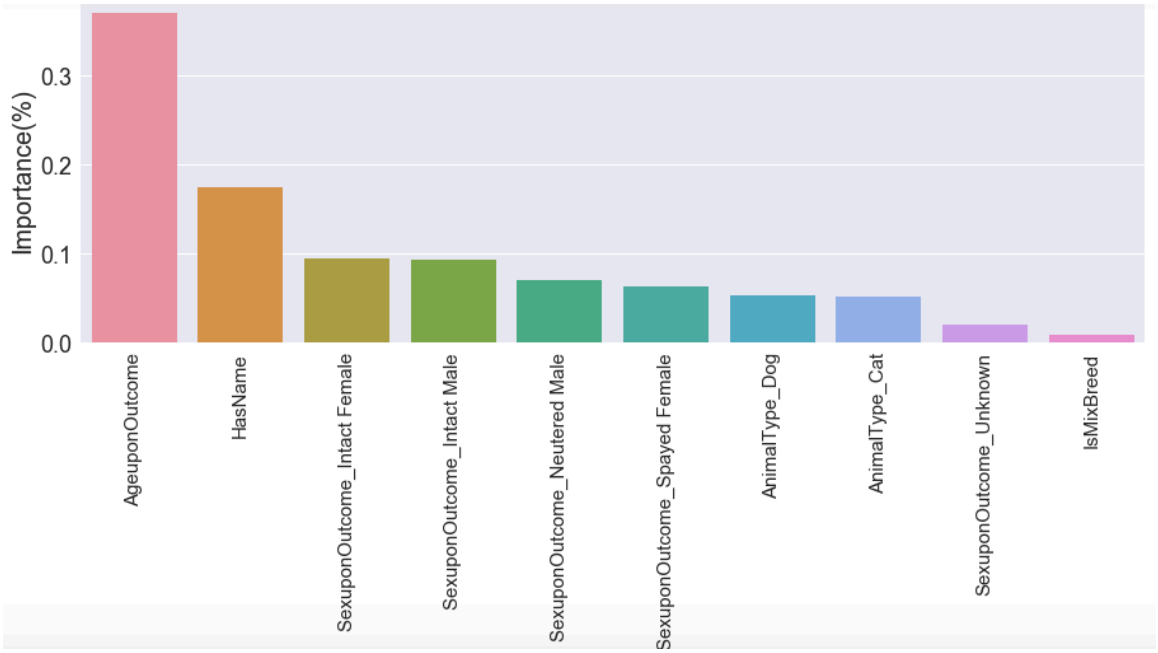
Justification

We made significant progress compared to the benchmark, and with parameter tuning was able to get as high as 0.82. The Kaggle leaderboard is ~0.65 for models that don't use the data leak. So we're not far from the best models. Also with the train and test loss being virtually identical, I think the model is robust as there is no sign of overfitting, i.e. it is generalizing well.

V. Conclusion

Free-Form Visualization

Let's look at the feature importance to get an idea of which features are the most useful for the model.



We see the age had the biggest impact (understandable, as puppies are more likely to be adopted than old dogs, for example), followed by whether or not the animal has a name (we saw this in exploratory analysis). However, we one-hot encoded the SexuponOutcome column that got split into 5 different columns, which if we add up their importance, would be roughly similarly important as the AgeuponOutcome. We saw that neutered and spayed animals are more likely to get good outcomes. IsMixBreed isn't very important; we perhaps needed to be cleverer in how we handled this column's data to get more use out of it.

Reflection

- Exploratory analysis and visualizations were helpful to understand the data and already give insights into which features are important for predicting outcome. E.g. having a name and being neutered/spayed would be advantageous for the animal's outcome.
- Needed to clean up some data such as age etc. (feature engineering), as well as data processing for use by the models. Had difficulty with Breed and Color, ended up not using them in the prediction.
- Ran a dummy model to get a benchmark, followed by training 2 proper models using grid search for parameter tuning, which beat the benchmark and got pretty good scores. We evaluated the models using the log loss metric and finally displayed the feature importance of the data.
- Some interesting aspects: The exploratory analysis to see what sort of values we have for the different features, including some which had many

null values that had to be dealt with (the Name column, which actually let us engineer a new feature of whether or not the animal has a name, as it could potentially affect the outcome), some visualization already gave us clues as to which features are going to be important, e.g. it's important to have a name and be neutered/spayed to have a higher chance of adoption.

- Some difficult aspects: The columns Breed and Color were tough to deal with, as there were too many unique values. I tried simplifying the Breed column, but as the feature importance shows, it did not prove too useful for prediction. Parameter tuning for XGBoost was also very slow, and I ended up having to abandon that. I think the final model could be used by the shelter to estimate the likelihood of good/bad outcomes and take action accordingly, for example if an animal has a lower chance of being adopted, maybe they could reduce the adoption fee for that animal or offer other incentives.

Improvement

- We could make more use of some of the features e.g. Perhaps group the dog breeds into dog groups such as ‘toy’, ‘hound’ etc., or extracting more info about the dogs from the breeds, such as their size, weight, intelligence, aggressiveness etc. These could be useful features in helping predict the outcome. We would need to use external data based on the dog breed to perform this analysis. Similar for the Color variable, grouping different colors into ‘Light’, ‘Medium’, ‘Dark’, for example to reduce the number of unique values in the column.
- We could try more parameter tuning on a more powerful machine with more memory, then perhaps the XGBoost could have performed better. Perhaps we could also try linear classifiers like SVM.