Machine Learning Engineer Nanodegree

Capstone Project
Ben Griffith
August 29, 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 shelter animals are euthanized in the US every year.[1]

Related datasets come from the Austin Animal Center for the time period October 1, 2013 to March, 2016. During intake, all animals receive a unique Animal ID and other information such as animal, sex, age, color and breed are logged. Outcomes represent the status of animals as they leave the Animal Center. Possible outcomes are adoption, died, euthanasia, return to owner and transfer.

Problem Statement
Approximately 2.7 million shelter animals are euthanized in the US every year.

In this multi-class classification problem, a dataset of intake information (breed, color, sex, age, etc.) provided by the Austin Animal Center will be used to train a supervised learning algorithm. The trained model will then be utilized to help predict the outcome (adoption, died, euthanasia, return to owner or transfer) of future shelter animals.

Knowing the predicted outcomes can help shelters identify and understand trends in animal outcomes. Such insights could help shelters focus their resources on specific animals who might need extra help finding a new home. For example, if the predicted outcome for a certain animal or breed in a shelter is euthanasia, the shelter could align their efforts to help see these euthanasia candidates find a new home.

Metrics
I plan to use accuracy as a metric for evaluating algorithms on the training data and test data. In addition, I plan to use F1 Score as a metric on the test data.

Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric used for classification problems.

---

[1] Shelter animal statistics were taken from The American Society for the Prevention of Cruelty to Animals (https://www.aspca.org/animal-homelessness/shelter-intake-and-surrender/pet-statistics)

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

The F1 Score is the Harmonic Mean between precision and recall. It calculates the weighted average of precision and recall. The F1 Score takes both false positives and false negatives into account. The greater the F1 Score means better performance of the model. Usually, the F1 Score is more useful if there is an uneven class distribution.

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

II. Analysis

Data Exploration
The training dataset (train.csv) provided by the Austin Animal Center contains 26,729 rows and 10 columns.

8 of the 10 columns are independent variables and the other 2 columns are being treated as the dependent variable.

Independent Variables:
- AnimalID - unique identifier
- Name - name of animal
- DateTime - date/time the animal was admitted to the animal shelter
- AnimalType - type of animal
- SexuponOutcome - sex of animal when animal left the animal shelter
- AgeuponOutcome - age of animal when animal left the animal shelter
- Breed - breed of animal
- Color - color of animal

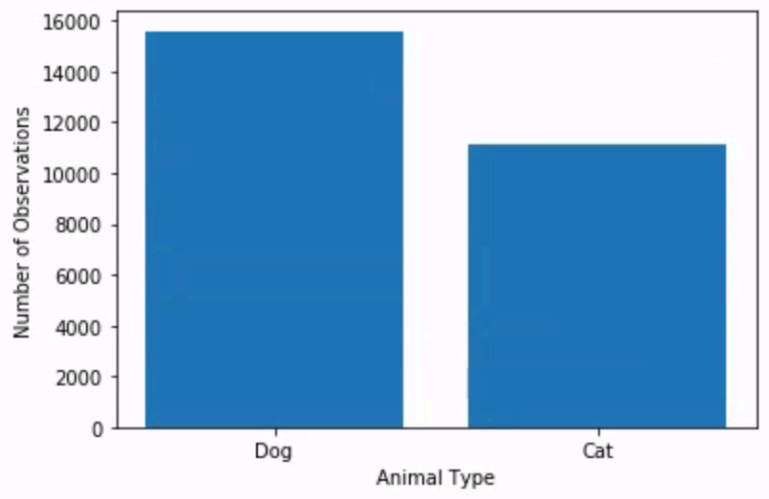
Dependent Variable(s):
- OutcomeType - ultimate outcome of each animal (Adoption, Died, Euthanasia, Return to Owner or Transfer)
- OutcomeSubtype

Dataset abnormalities to be addressed:
- All independent variables are categorical

- Name, OutcomeSubtype, SexuponOutcome and AgeuponOutcome contain missing values
- AgeuponOutcome, Breed and Color contain a significant amount of distinct values
  - For example, AgeuponOutcome contains 1 year, 2 years, 2 months, 3 years, 1 month, 3 months, 4 years, 5 years, 4 months, 6 years, 1 day, 4 days, 5 days and many others
  - Breed contains Domestic Shorthair Mix, Pit Bull Mix, Chihuahua Shorthair Mix, Labrador retriever Mix, German Shepherd Mix, Domestic Longhair Mix, Siamese Mix and many others
  - Color contains Black/White, Black, Brown Tabby, Brown Tabby/White, White, Brown/White, Orange Tabby, Tan/White, Tricolor, Blue/White, Black/Tan, White/Black, Brown and many others

Data Visualization

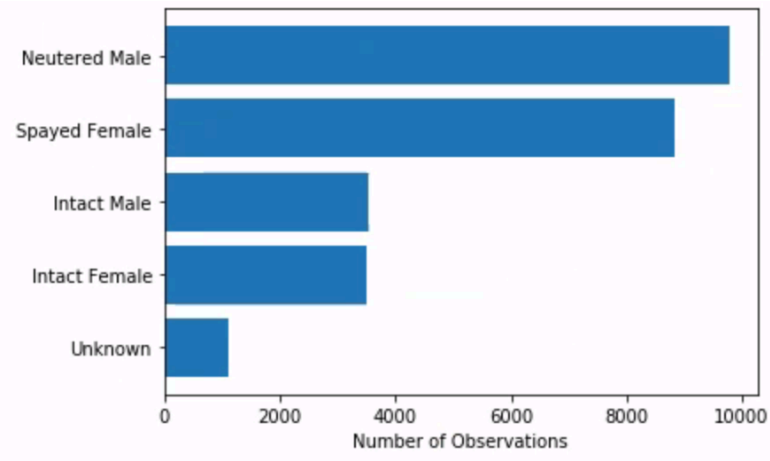
These plots show how the dataset is split for two relevant independent variables: AnimalType and SexuponOutcome.



The total number of observations in this dataset is 26,729.

Regarding AnimalType, 58.3% of observations are Dog and 41.7% of observations are Cat.

Regarding SexuponOutcome, 36.6% of observations are Neutered Male, 33.1% of observations are Spayed Female, 13.2% of observations are Intact Male, 13.1% of observations are Intact Female and 4% of observations Unknown.



Algorithms and Techniques
First, using train test split, 65% of the dataset will be used to fit each algorithm while 35% of the dataset will be used to test on the final and benchmark models.

Using k-fold cross validation, multi-class classification algorithms will be fit to the training dataset. For k-fold cross validation, a value of 5 will be assigned for k. The algorithms I plan to use are:

- Decision Tree (default parameters)
- K Neighbors (default parameters)
- XGBoost (default parameters)
- Naive Bayes (default parameters)
- Support Vector Machines (default parameters)

The accuracy for each model will be compared and the parameters for model with the highest accuracy will then be tuned using GridSearchCV.

Ensemble algorithms will be fit to the training dataset and, like earlier, the accuracy for each model will be compared to the tuned model. The best performing model will be used on the unseen data. The Ensemble algorithms I plan to use are:

- AdaBoost (default parameters)
- Gradient Boosting (default parameters)
- Random Forest (default parameters)
- Extra Trees (default parameters)

Benchmark
As the benchmark model, the Logistic Regression classifier will be fit to the training data and make prediction on unseen data.

Data	Training Accuracy	Test Accuracy
Cat	72.3%	71.5%
Dog	56.2%	57.6%

III. Methodology

Data Preprocessing
Due to the nature and size of the dataset, I decided to split the original dataset into two segments: Cat and Dog.

For the Cat dataset and Dog dataset, the following preprocessing steps were implemented:

1. Created a Cat data frame and Dog data frame by copying the original data frame.
2. Removed all Dog observations from the Cat data frame. Removed all Cat observations from the Dog data frame.
3. Removed any observation where NaN was present in AgeuponOutcome and SexuponOutcome features for both data frames.
4. Removed observations from SexuponOutcome feature that had a value of ‘Unknown’ for both data frames.
5. Cat and Dog observations that had a name were updated to the value of ‘Known’.
6. Cat and Dog observations that did not have a name (NaN) were updated to the value of ‘Unknown’.
7. Split Cat data frame and Dog data frame into independent variables and dependent variable.
8. Converted AgeuponOutcome to a numerical value (number of days) for both data frames.
9. Strings ‘Mix’ and ‘/‘ were removed from Breed and Color independent variables for both data frames.
10. Color observations value updated to ‘Mixed’ or ‘Non-mixed’ for both data frames. If the Color value contains ‘/‘, then the value is updated to ‘Mixed’.
11. Breed observations below a threshold of 50 were updated to ‘Other’ for both data frames.
12. Color observations below a threshold of 50 were updated to ‘Other’ for both data frames.
13. AgeuponOutcome was scaled to a numerical value for both data frames.
14. Implemented one-hot encoding for categorical features in both data frames.

Implementation

In util.py, I created three functions: convertAgeToDays(), getBreed() and getColor(). These functions were used to help preprocess data.

```
def convertAgeToDays(age):
    '''
    Converting value from AgeuponOutcome to age in number of days
    '''

    days_in_week = 7
    days_in_month = 30
    days_in_year = 365
    age_list = age.split()

    if age_list[1][0] == 'w':
        age_in_days = int(age_list[0]) * 7
    elif age_list[1][0] == 'm':
        age_in_days = int(age_list[0]) * 30
    elif age_list[1][0] == 'y':
        age_in_days = int(age_list[0]) * 365
    else:
        age_in_days = int(age_list[0])

    return age_in_days
```

convertAgeToDays() accepts one argument, age, and converts it to the number of days.

For example:

age	age_in_days
1 week	7
3 months	90
2 years	760
2 days	2

```
def getBreed(animal_breed):
    '''
    getBreed takes in one argument and if 'Mix' or '/' is found these values will be removed or replaced with whitespace
    '''
    breed_name = ''
    breed = animal_breed.split()
    slash_exist = False

    if 'Mix' in breed:
        breed.pop()
        breed_name = ' '.join(breed)
    else:
        breed = ' '.join(breed)
        for i in breed:
            if i == '/' :
                slash_exist = True

        if slash_exist:
            breed_name = breed.replace('/', ' ')
        else:
            breed_name = breed

    return breed_name
```

getBreed() accepts one argument, animal_breed, and if it contains ‘Mix’ or ‘/‘, the returned value will be altered so that ‘Mix’ or ‘/‘ has been removed or replaced. For example:

For example:

animal_breed	breed_name
Domestic Shorthair Mix	Domestic Shorthair
Siamese Mix	Siamese
Boston Terrier/Beagle	Boston Terrier Beagle
Rat Terrier/Pointer	Rat Terrier Pointer

```
def getColor(animal_color):
    '''
    getColor takes in animal_color and returns one of two values based on whether a '/' is found
    in the argument
    '''

    for i in animal_color:
        if i == '/':
            color = 'Mixed'
            break
        else:
            color = 'Non-Mixed'

    return color
```

getColor() accepts one argument, animal_color, and if it contains '/', the returned value will be 'Mixed' otherwise the returned value will be 'Non-Mixed'.

For example:

animal_color	Color
Black/White	Mixed
Brown	Non-Mixed

After the data has been preprocessed, the train_test_split function is used to split the Cat and Dog datasets into training data and test data. The test data size used is 35%.

```
# Split into train and test set
cat_x_train, cat_x_test, cat_y_train, cat_y_test = train_test_split(cat_final, cat_y, test_size=0.35, random_state=42)
```

```
# Split into train and test set
dog_x_train, dog_x_test, dog_y_train, dog_y_test = train_test_split(dog_final, dog_y, test_size=0.35, random_state=42)
```

```
# Spot-check algorithms for Cat data set
models = []
models.append(('LG', LogisticRegression(solver='liblinear', multi_class='ovr'))) # Benchmark model
models.append(('CART', DecisionTreeClassifier()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('XGB', XGBClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

for name, model in models:
    kfold = KFold(n_splits=5, random_state=42)
    cv_results = cross_val_score(model, cat_x_train, cat_y_train.values.ravel(), cv=kfold, scoring='accuracy')
    print('{}: {}'.format(name, cv_results.mean()))
```

```
LG: 0.7232053658474382
CART: 0.7198548411216252
KNN: 0.7050853135893147
XGB: 0.7423869257315616
NB: 0.3045065096604497
SVM: 0.7209221449015292
```

Model training and selection for the Cat dataset follows the sequence below:

1. A list of tuples, models, is created and contains the following algorithms:
   - LogisticRegression()
   - DecisionTreeClassifier()
   - KNeighborsClassifier()
   - XGBClassifier()
   - GaussianNB()
   - SVC()
2. With k-fold cross validation, each model is fit to the training data and evaluated using the accuracy metric. The number of folds is 5.
3. At the end of each loop, the metric is printed out for the appropriate model.
4. The model with the highest accuracy score will be used in the refinement process.

As seen above, the XGBClassifier reported the highest accuracy for the Cat dataset.

```
# Spot-check algorithms for Dog data set
models = []
models.append(('LG', LogisticRegression(solver='liblinear', multi_class='ovr'))) # Benchmark model
models.append(('CART', DecisionTreeClassifier()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('XGB', XGBClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

for name, model in models:
    kfold = KFold(n_splits=5, random_state=42)
    cv_results = cross_val_score(model, dog_x_train, dog_y_train.values.ravel(), cv=kfold, scoring='accuracy')
    print('{}: {}'.format(name, cv_results.mean()))
```

```
LG: 0.5621219633118492
CART: 0.5197818542389687
KNN: 0.5186911254338126
XGB: 0.5685671789786813
NB: 0.07694595934556273
SVM: 0.5370352007932573
```

Model training and selection for the Dog dataset follows the sequence below:

1. A list of tuples, models, is created and contains the following algorithms:
   - LogisticRegression()
   - DecisionTreeClassifier()
   - KNeighborsClassifier()
   - XGBClassifier()
   - GaussianNB()
   - SVC()
2. With k-fold cross validation, each model is fit to the training data and evaluated using the accuracy metric. The number of folds is 5.
3. At the end of each loop, the metric is printed out for the appropriate model.
4. The model with the highest accuracy score will be used in the refinement process.

As seen above, the XGBClassifier reported the highest accuracy for the Dog dataset.

Refinement

The refinement process consisted of the following:

1. Hyper parameter tuning
2. Ensemble model comparison against the tuned model

For the Cat dataset, the initial solution was an XGBClassifier, which reported accuracy around 74.2%.

```
# Cat
learning_rate = [0.05, 0.10, 0.15]
min_child_weight = [1, 3, 5]
gamma = [0.0, 0.1, 0.2, 0.3]
sample = [0.25, 0.50, 0.75]
param_grid = dict(learning_rate=learning_rate, min_child_weight=min_child_weight, subsample=sample, gamma=gamma)
model = XGBClassifier()
kfold = KFold(n_splits=3, random_state=42)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=kfold, iid=True)
grid_result = grid.fit(cat_x_train, cat_y_train.values.ravel())

print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('%f, (%f) with: %r' % (mean, stdev, param))
```

```
Best: 0.744062 using {'gamma': 0.3, 'learning_rate': 0.05, 'min_child_weight': 3, 'subsample': 0.25}
```

GridSearchCV was used to complete hyper parameter tuning for the following parameters:

1. learning_rate
2. min_child_weight
3. gamma
4. subsample

As a result, accuracy increased by 0.02% to 74.4% for the tuned model.

For the Dog dataset, the initial solution was an XGBClassifier, which reported accuracy

```
# Dog
learning_rate = [0.05, 0.10, 0.15]
min_child_weight = [1, 3, 5]
gamma = [0.0, 0.1, 0.2, 0.3]
sample = [0.25, 0.50, 0.75]
param_grid = dict(learning_rate=learning_rate, min_child_weight=min_child_weight, subsample=sample, gamma=gamma)
model = XGBClassifier()
kfold = KFold(n_splits=3, random_state=42)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=kfold, iid=True)
grid_result = grid.fit(dog_x_train, dog_y_train.values.ravel())

print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('%f, (%f) with: %r' % (mean, stdev, param))
```

```
Best: 0.571740 using {'gamma': 0.1, 'learning_rate': 0.1, 'min_child_weight': 3, 'subsample': 0.75}
```

around 56.8%.

GridSearchCV was used to complete hyper parameter tuning for the following parameters:

1. learning_rate
2. min_child_weight
3. gamma
4. subsample

As a result, accuracy increased by 0.031% to 57.1% for the tuned model.

The accuracy for the tuned models was then compared with the accuracy of Ensemble methods. For both datasets, even though GradientBoostingClassifier was the highest performing Ensemble it did not outperform the tuned models.

```
# Ensembles for Cat dataset
ensembles = []
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
ensembles.append(('RF', RandomForestClassifier(n_estimators=10)))
ensembles.append(('ET', ExtraTreesClassifier(n_estimators=10)))

for name, model in ensembles:
    kfold = KFold(n_splits=5, random_state=42)
    cv_results = cross_val_score(model, cat_x_train, cat_y_train.values.ravel(), cv=kfold, scoring='accuracy')
    print('{} {}'.format(name, cv_results.mean()))
```

```
AB 0.6479923861722316
GBM 0.7390384876211542
RF 0.7238138460842922
ET 0.722290848684447
```

```
# Ensembles for Dog dataset
ensembles = []
ensembles.append(('AB', AdaBoostClassifier()))
ensembles.append(('GBM', GradientBoostingClassifier()))
ensembles.append(('RF', RandomForestClassifier(n_estimators=10)))
ensembles.append(('ET', ExtraTreesClassifier(n_estimators=10)))

for name, model in ensembles:
    kfold = KFold(n_splits=5, random_state=42)
    cv_results = cross_val_score(model, dog_x_train, dog_y_train.values.ravel(), cv=kfold, scoring='accuracy')
    print('{} {}'.format(name, cv_results.mean()))
```

```
AB 0.5458601883986118
GBM 0.5659890927119485
RF 0.5232523549826475
ET 0.5185919682697075
```

IV. Results

Model Evaluation and Validation

```
# Prepare the benchmark
print('Cat Benchmark----------------------------------------')
benchmark = LogisticRegression(solver='liblinear', multi_class='ovr')
benchmark.fit(cat_x_train, cat_y_train.values.ravel())

# Estimate accuracy on validation dataset
predictions = benchmark.predict(cat_x_test)
print('Accuracy:', accuracy_score(cat_y_test, predictions))
print('F1 Score:', f1_score(cat_y_test, predictions, average='weighted'))

# Prepare the model
print('Cat Model----------------------------------------')
model = XGBClassifier(gamma=0.3, learning_rate=0.05, min_child_weight=3, subsample=0.25)
model.fit(cat_x_train, cat_y_train.values.ravel())

# Estimate accuracy on validation dataset
predictions = model.predict(cat_x_test)
print('Accuracy:', accuracy_score(cat_y_test, predictions))
print('F1 Score:', f1_score(cat_y_test, predictions, average='weighted'))
```

```
# Prepare the benchmark
print('Dog Benchmark----------------------------------------')
benchmark = LogisticRegression(solver='liblinear', multi_class='ovr')
benchmark.fit(dog_x_train, dog_y_train.values.ravel())

# Estimate accuracy on validation dataset
predictions = benchmark.predict(dog_x_test)
print('Accuracy:', accuracy_score(dog_y_test, predictions))
print('F1 Score:', f1_score(dog_y_test, predictions, average='weighted'))

# Prepare the model
print('Dog Model----------------------------------------')
model = XGBClassifier(gamma=0.3, learning_rate=0.15, min_child_weight=1, subsample=0.75)
model.fit(dog_x_train, dog_y_train.values.ravel())

# Estimate accuracy on validation dataset
predictions = model.predict(dog_x_test)
print('Accuracy:', accuracy_score(dog_y_test, predictions))
print('F1 Score:', f1_score(dog_y_test, predictions, average='weighted'))
```

In model evaluation and validation, the benchmark model and tuned model were fit to the training data for each dataset (Cat and Dog) and then predictions were made on the unseen, test data. Accuracy and the F1 Score were used to evaluate the performance of each model's predictions.

As stated in the previous section, the final model used for the Cat dataset and Dog dataset was a tuned XGBoost model. Using GridSearchCV to evaluate a high number of parameter combinations, the final parameters and the selected values are appropriate (as seen below).

Parameter	Tuned Model for Cat	Tuned Model for Dog
gamma	0.3	0.3
learning_rate	0.05	0.15
min_child_weight	3	1
subsample	0.25	0.75

As seen in the metrics below, even though both tuned models outperform the benchmark models, I think only the tuned model for the Cat dataset can be trusted as it appears to generalize well on unseen data and perform relatively satisfactory. While the metrics on unseen data were similar to the training data for the Dog dataset, the accuracy and F1 Score were significantly lower than the Cat dataset so I don't think the tuned model from the Dog dataset can be trusted.

Model	Accuracy (Cat)	F1 Score (Cat)	Accuracy (Dog)	F1 Score (Dog)
Benchmark (Logistic Regression)	71.6%	67.7%	57.6%	54.1%
Tuned Model (XGBoost)	74.3%	70.6%	58.2%	56.0%

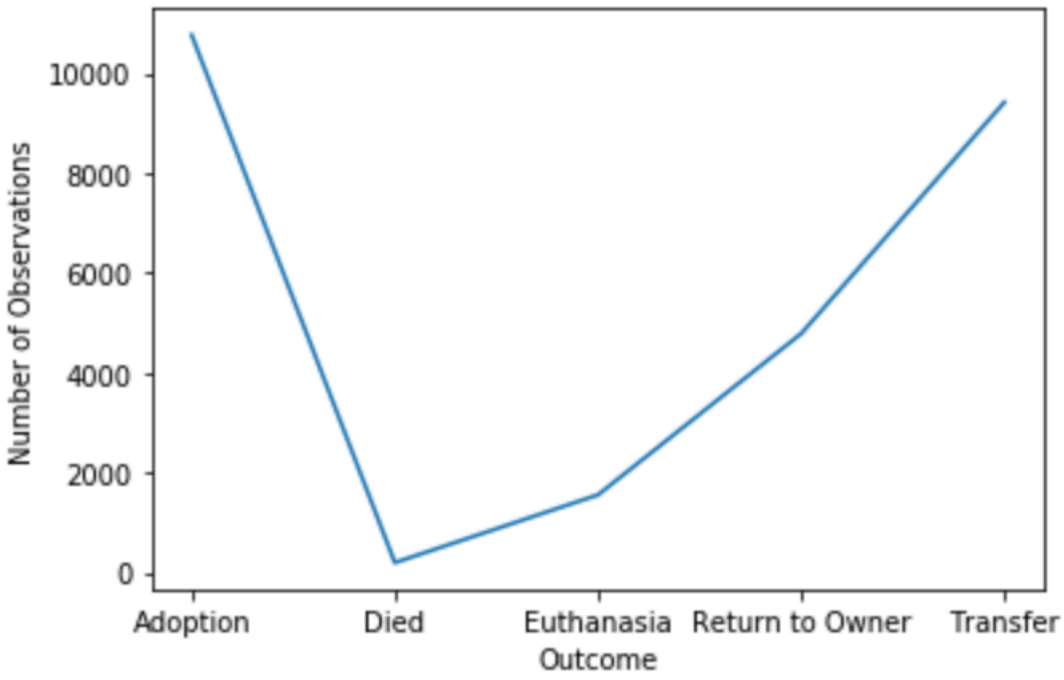
For both datasets (Cat and Dog), the tuned XGBoost Classifier outperformed the Logistic Regression benchmark in terms of accuracy and the F1 Score. Both the benchmark and tuned model appeared to do better on the Cat dataset than the Dog dataset. For example, the accuracy for the tuned model for the Cat dataset was 74.3% and for the Dog dataset was 58.2% roughly a 16% difference and for the F1 Score the difference was roughly 14%.

While the tuned Dog model was able to outperform the benchmark, I don't think the model is significant enough to have solved the problem. The tuned Cat model, on the other hand, comes closest to a final solution for the problem.

V. Conclusion

Free-Form Visualization

The solution to the problem of 2.7 million animals in shelters being euthanized each year centered around being able to make predictions on the outcome of each shelter animal based on a few characteristics. During exploratory data analysis, I found it interesting the dataset had only roughly 6% euthanasia outcomes. Although the euthanasia number of observations is low for this particular dataset it does mean the other outcomes are high such as adoption, return to owner or transfer. Knowing the characteristics of those animals that are adopted, etc. might translate into shelters having more insight into which animals are not adopted and more likely to face euthanasia.



Outcome	# of Observations
Adoption	10,769
Died	197
Euthanasia	1,555
Return to Owner	4,786
Transfer	9,422

Reflection
Below is the end-to-end process I used for this project:

I started by loading the libraries and dataset. I then summarized the data using descriptive statistics such as .info(), .describe(), .head(), .value_counts() and .shape. I also used data visualization for some basic plots to display different feature characteristics. After data summarization, I preprocessed the data. Next, the datasets were split into training and test data and the algorithms were fit to the training data. Using accuracy, the best performing algorithm was selected to refine. During model improvement, hyper parameter tuning was conducted using GridSearchCV to tune the model. The tuned model was compared against Ensemble methods. The tuned model was then evaluated on unseen data.

The part of the project I found most challenging was the data preprocessing. I started with one dataset, but after conducting exploratory data analysis I decided to split up the datasets by animal type. I ended up creating functions to handle animal age, breed and color. Going through the process of data cleaning made me realize how important it is to have preprocessed data before training models.

Improvement
In this project, I think the process or techniques used in data preprocessing could be improved. As I mentioned earlier, I found data cleaning to be the most challenging task of this project and, while I feel I have done the best I can do, I am sure there are better ways to handle the data cleaning process.

Regarding alternative algorithms, I researched Light GBM, but did not use it because I decided to implement XGBoost instead. Maybe implementing Light GBM would have resulted in a better outcome.