

Udacity MLE Nanodegree Capstone

Project Background

Problem Statement:

There is a plethora of animals in shelters waiting for adoption and rescue from people in communities. To better understand the trends and gaps in data about these animals, a dataset will be studied, and a model trained to predict the probability of animals being adopted within the allowable timeframe of retention. Predicting shelter animal outcomes will then allow this project to generate ideas for animal shelters to improve processes, operations, and strategies for animals during the permitted allowable time they are in the shelter to aid in reducing suffering and non-adoption of these animals.

Domain Background:

Open Data Initiatives around the world are allowing government and individuals to innovate and collaborate in new ways. Open Data provides the opportunity for transparency in communities, public service improvement, innovation, and economic value. For more information about Open Data Initiatives and resources, please visit: <http://opendatatoolkit.worldbank.org/en/starting.html>. The Open Data domain of study chosen is Animal Welfare and operations at animal shelters.

Solution Methodology:

- Exploratory Data Analysis (EDA): that dataset will be explored using tools demonstrated in this course, including Pandas python package, Auto-visualization tools
- Data Cleaning: Following exploratory data analysis, the data will be cleaned if there are missing values, inconsistencies in data, or imbalances. Further, feature engineering may be required for categories of outcomes and animals.
- Model Selection: Since the data is tabular and has a mix of data types, the most likely models challenged against each other will be XGBoost-related and ensemble learning techniques. An AutoML tool may be used to expedite comparison of model options.
- Model Hyperparameter Optimization: Once models are compared and a champion is selected, the model hyperparameters will be optimized using AWS Sagemaker tools.
- Model Training: From hyperparameter optimization, the best hyperparameters will be extracted and used for final training of model predicting outcomes of animals based on features.
- Model Deployment: Finally, the model will be deployed to an endpoint and tested to ensure the response is understandable.

Deliverables:

AWS Sagemaker Notebook Outlining Process Findings

- EDA Visualizations and Summary
- Machine Learning Model Testing and Training
- Sample Code for Deployment

Blog-ready Documentation about Model and Analysis Key Takeaways

- Benchmark Model Comparative Analysis

- List of possible actionable takeaways for operations and stakeholders

Project Setup

- Dataset: The dataset used for this project was introduced by the Kaggle platform. There are a total of three tabular datasets with 9 to 10 features that will be analyzed and down selected from to train a machine learning model.
- Platform(s): Google Co-lab free resources will be used for exploration and experimentation and final deliverables and notes will be transitioned to AWS Sagemaker Notebooks for documentation and final product integration purposes.
- Dependencies/Packages:
- Metrics: F1 score to determine validity of model and binary log loss to monitor training

Solution Process:

1. Exploratory Data Analysis

For initial exploration, [Google Co-lab](#), a free resource introduced by the Machine Learning Engineering Nanodegree course, that can run notebooks similar to AWS Sagemaker was used to look at basic characteristics of the data.

The first exploration was looking at the column names or features per file. In order to review features and compare, pandas python package and the info() function was used. Here are the results:

'acc_intakes.csv':

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80187 entries, 0 to 80186
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  --
0   age_upon_intake        80187 non-null  object
1   animal_id              80187 non-null  object
2   animal_type            80187 non-null  object
3   breed                  80187 non-null  object
4   color                  80187 non-null  object
5   datetime               80187 non-null  object
6   datetime2              80187 non-null  object
7   found_location         80187 non-null  object
8   intake_condition       80187 non-null  object
9   intake_type            80187 non-null  object
10  name                   55603 non-null  object
11  sex_upon_intake        80186 non-null  object
dtypes: object(12)
memory usage: 7.3+ MB
```

'aac_outcomes.csv':

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80681 entries, 0 to 80680
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  --
0   age_upon_outcome       80673 non-null  object
1   animal_id              80681 non-null  object
2   animal_type            80681 non-null  object
3   breed                  80681 non-null  object
4   color                  80681 non-null  object
5   date_of_birth          80681 non-null  object
6   datetime               80681 non-null  object
7   monthyear              80681 non-null  object
8   name                   56116 non-null  object
9   outcome_subtype        36893 non-null  object
10  outcome_type            80667 non-null  object
11  sex_upon_outcome       80679 non-null  object
dtypes: object(12)
memory usage: 7.4+ MB
```

There were two datasets that described the intake and outcome characteristics per animal. There were some redundant features, such as animal identification, animal type, breed, and color.

The largest dataset had combined columns from both the intake and outcome files:

'acc_intakes_outcomes.csv':

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79672 entries, 0 to 79671
Data columns (total 41 columns):
#   Column                Non-Null Count  Dtype
---  --
0   age_upon_outcome       79672 non-null  object
1   animal_id_outcome      79672 non-null  object
2   date_of_birth          79672 non-null  object
3   outcome_subtype        36348 non-null  object
4   outcome_type           79662 non-null  object
5   sex_upon_outcome       79671 non-null  object
6   age_upon_outcome (days) 79672 non-null  int64
7   age_upon_outcome (years) 79672 non-null  float64
8   age_upon_outcome_age_group 79672 non-null  object
9   outcome_datetime       79672 non-null  object
10  outcome_month          79672 non-null  int64
11  outcome_year           79672 non-null  int64
12  outcome_monthyear      79672 non-null  object
13  outcome_weekday        79672 non-null  object
14  outcome_hour           79672 non-null  int64
15  outcome_number         79672 non-null  float64
16  dob_year               79672 non-null  int64
17  dob_month              79672 non-null  int64
18  dob_monthyear          79672 non-null  object
19  age_upon_intake        79672 non-null  object
20  animal_id_intake       79672 non-null  object
```

```
21  animal_type            79672 non-null  object
22  breed                  79672 non-null  object
23  color                  79672 non-null  object
24  found_location         79672 non-null  object
25  intake_condition       79672 non-null  object
26  intake_type            79672 non-null  object
27  sex_upon_intake        79671 non-null  object
28  count                  79672 non-null  int64
29  age_upon_intake (days) 79672 non-null  int64
30  age_upon_intake (years) 79672 non-null  float64
31  age_upon_intake_age_group 79672 non-null  object
32  intake_datetime        79672 non-null  object
33  intake_month           79672 non-null  int64
34  intake_year            79672 non-null  int64
35  intake_monthyear       79672 non-null  object
36  intake_weekday         79672 non-null  object
37  intake_hour            79672 non-null  int64
38  intake_number          79672 non-null  float64
39  time_in_shelter        79672 non-null  object
40  time_in_shelter_days   79672 non-null  float64
dtypes: float64(5), int64(11), object(25)
memory usage: 24.9+ MB
```

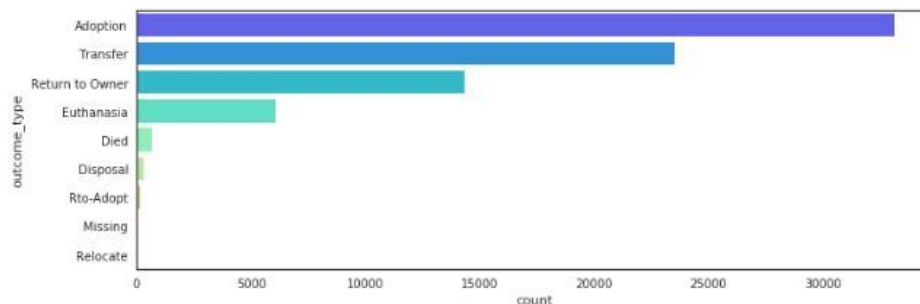
Notice the inconsistent record entries per file. Although, the largest file, 'acc_intakes_outcomes.csv', combined column names from the other two files and had additional features, there were fewer total entries in the file.

With the file combination, it became evident that the file with both features could be used for model training, and the intake information file could use to test the results of the model training, and, finally, measured with the outcomes information file to see how many times the model could predict the outcome correctly. The feature the datasets would be used to create a model that could predict the 'outcome_type' feature from the other input features. The 'outcome_types' in the dataset included these categories: 'Return to Owner', 'Transfer', 'Adoption', 'Euthanasia', 'Died', 'Rto-Adopt', 'Missing', 'Disposal', 'Relocate'. For the purposes of this project to simplify the process of training a model, the outcome would be categorized into two classes: adopted vs. not adopted, within the retention of the animal at the facility. This would then help the model solve for a binary classification problem.

The next step was to clean the data files and ensure consistency across datasets and create some visualizations that will help describe initial observed trends and correlations. The large dataset, 'acc_intakes_outcomes.csv', was focused on for this step.

From the benchmark model documentation, the categories for the model outcomes were evidently imbalanced. The dataset would need to be balanced to help the model have enough data to generate a better prediction. To balance the data, the outcome was re-grouped into 'adopted' and 'not adopted' categories.

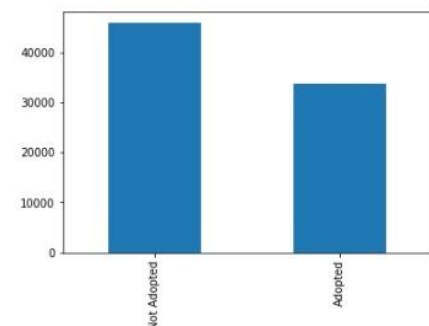
Here is the benchmark count of the data outcome types:



Sourced from: <https://www.kaggle.com/code/wenlie/exploring-and-predicting-the-animal-s-outcomes>

Once all the outcome types were re-categorized into the two primary categories, 'adopted' and 'not adopted', the data was better balanced for model training:

```
Not Adopted    0.576046
Adopted        0.423954
Name: outcome_type, dtype: float64
```



Further, the data types were modified to better match the intended input to the model. The columns that contained repeating values were changed to the 'category' data type. While datetime columns were

changed to the 'datetime64[ns]' data type. Columns were dropped based on having too missing values or redundant values, or repeating information, that already existed in other columns.

The full analysis and data exploration can be observed in the 'capstone_eda.ipynb' file. Bar charts and histograms were plotted to view distributions and counts for the dataset. Further, the 'pandas_profiling_report.html' files contain the correlation matrices and descriptive statistics of the datasets before and after the dataset was cleaned. Snapshots of each summary are included in Appendix A of this report. The cleaned dataset was then exported as a csv and used for the rest of development.

2. Model Selection

To kickoff model development, the AutoGluon template provided in the Machine Learning Engineering Nanodegree program was used to run AutoML AutoGluon package to test multiple models simultaneously with the cleaned dataset. The AutoML tool was ran using metrics, mean squared error, accuracy, and f1 score, that assisted with model selection and evaluation.

The full analysis and data exploration can be observed in the 'capstone_model_selection.ipynb' file. The cleaned dataset was split for training and testing, rows with missing values dropped, and a summary of the model performance based on specified metrics. The top performing models were Weighted Ensemble, Light GBM, Random Forest, and CatBoost:

Model Name	Root Mean Squared Error	Accuracy	F1 Score
Light GBM	-0.34	0.88	0.88
Random Forest	-0.35	0.876	0.87
CatBoost	-0.35	0.874	0.86

Each tabular predictor fit function was given the same amount of time, 600 second time limit, and preset of best quality to determine top models on specified metrics. The best scoring models seem to be decision tree-based models, gradient boost machine (GBM) and random forest model. This is different than initial thought, but like the decisions made in benchmark model.

3. Model Hyperparameter Optimization

At the start of hyperparameter optimization of the model, general advising was sought out during the last connect session for the course. Starting on this step, the following advice was kept in mind:

“When build model; make sure model is not overfitting; tendency of tree-based models is that they are overfit; can prune trees. Regularize parameters; make sure model performs good on training and test; if see big difference, reduce number of parameters of model, or increase dataset if can.” – Jason Zeng, Connect Session Lead, 03/17/23

To begin understanding how to optimize the model hyperparameters, the source and repo for the LightGBM algorithm was studied from a [Microsoft Github repository](#). Further, official [AWS documentation](#) reviews hyperparameters, definitions, objective metrics, and thresholds. Based on these studies, it was determine that the objective metric that needed to be used to measure the model training hyperparameters was a metric known as binary log loss, since the problem presents a binary classification problem.

After reading the documentation, a built-in algorithm in AWS was selected to optimize and hyperparameter tune. To use the built-in algorithm, the dataset had to be further modified to meet the specifications of the docker image.

After extensive reading about the AWS Sagemaker 'transfer_learning.py' file for the Sagemaker estimator, there were unknown errors experienced while running hyperparameter tuning for the pre-developed light gradient boost machine algorithm. After further investigation, the SciKitLearn version was studied and used instead in the corresponding AWS Sagemaker notebook.

Further, the ChatGPT tool introduced in the AWS Machine Learning nanodegree course was used to provide a simple example of hyperparameter tuning using the SciKitLearn "GBMClassifier" function. The AUC ROC metric was used to measure the best hyperparameters from hyperparameter values input for the model. The hyperparameters tuned are listed in the tables with either values tuned, or ranges selected.

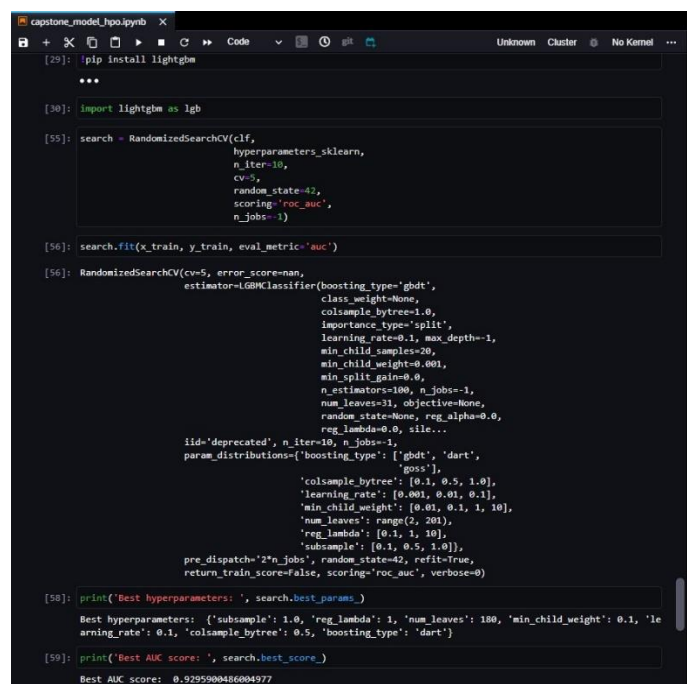
Categorical Hyperparameters:				
Boosting type	Gbdt	Dart	Goss	
Integer Hyperparameter Values:				
Learning rate	0.001	0.01	0.1	
Minimum child weight	0.01	0.1	1	10
Subsample	0.1	0.5	1.0	
Column sample by tree	0.1	0.5	1.0	
Regular lambda	0.1	1	10	
Hyperparameter Continuous Ranges:				
Number of leaves	2	201		

The successful attempt at hyperparameter tuning results may be reviewed in a screenshot included in this report.

The best hyperparameter tuning values outputted received an AUC ROC score of 0.93.

The hyperparameter values were then saved and used for the next section of the model development process. Tested against validation data, the benchmark accuracy kept in mind for model training was around 0.85.

Lastly, feature importance was studied from hyperparameter optimization, and it was evident that the weekday of adoption and age upon intake were the features with the highest importance. The intake condition of each animal also seemed to be significant.



```
[29]: !pip install lightgbm
...

[30]: import lightgbm as lgb

[55]: search = RandomizedSearchCV(clf,
                                hyperparameters_sklarn,
                                n_iter=10,
                                cv=5,
                                random_state=42,
                                scoring='roc_auc',
                                n_jobs=-1)

[56]: search.fit(x_train, y_train, eval_metric='auc')

[56]: RandomizedSearchCV(cv=5, error_score=nan,
                        estimator=LGBMClassifier(boosting_type='gbdt',
                                                class_weight=None,
                                                colsample_bytree=1.0,
                                                importance_type='split',
                                                learning_rate=0.1, max_depth=-1,
                                                min_child_samples=20,
                                                min_child_weight=0.001,
                                                min_split_gain=0.0,
                                                n_estimators=100, n_jobs=-1,
                                                num_leaves=31, objective=None,
                                                random_state=None, reg_alpha=0.0,
                                                reg_lambda=0.0, silent=True,
                                                iid='deprecated', n_iter=10, n_jobs=-1,
                                                param_distributions={'boosting_type': ['gbdt', 'dart',
                                                'goss'],
                                                'colsample_bytree': [0.1, 0.5, 1.0],
                                                'learning_rate': [0.001, 0.01, 0.1],
                                                'min_child_weight': [0.01, 0.1, 1, 10],
                                                'num_leaves': range(2, 201),
                                                'reg_lambda': [0.1, 1, 10],
                                                'subsample': [0.1, 0.5, 1.0]}},
                        pre_dispatch='2*n_jobs', random_state=42, refit=True,
                        return_train_score=False, scoring='roc_auc', verbose=0)

[58]: print("Best hyperparameters: ", search.best_params_)

Best hyperparameters: {'subsample': 1.0, 'reg_lambda': 1, 'num_leaves': 100, 'min_child_weight': 0.1, 'learning_rate': 0.1, 'colsample_bytree': 0.5, 'boosting_type': 'dart'}

[59]: print("Best AUC score: ", search.best_score_)

Best AUC score: 0.9295900486004977
```



```
[35]: print(classification_report(y_val, clf_predict))
```

	precision	recall	f1-score	support
0	0.82	0.83	0.82	6716
1	0.87	0.86	0.87	9217
accuracy			0.85	15933
macro avg	0.84	0.85	0.84	15933
weighted avg	0.85	0.85	0.85	15933

The F1 score calculated for the validation dataset is considered acceptable, however, given precautionary insight about tree models and overfitting, more information is extracted during training the model to ensure that the algorithm is not memorizing information.

4. Model Training

Using the LGBMClassifier from lightgbm, I was able to specify the best hyperparameters found during hyperparameter tuning. During training, I decided to log the loss of the model during training to visually search for evidence of underfitting, generalization, and overtraining. After training and plotting the loss, the loss steadily decreased and did not show significant separation between the loss during training and validation. A graph of the metric logs during training from the AWS Sagemaker notebooks may be found in Appendix B.

Throughout the notebooks generated for hyperparameter optimization and training, the dataset used was encoded using various methods for the model to be able to interpret. In order to understand how this could affect feature importance, feature importance was plotted for hyperparameter optimization and training. The expectation was that feature importance would remain unchanged or similar in both cases, however, the results were significantly different.

During hyperparameter optimization, the outcome time features appeared to have more importance while animal and intake characteristics appeared to have more importance during training. This result indicates that there are different branches in the hyperparameter optimization modeling and final model using the best hyperparameters. Although, outcome timing represents features that are, generally, at the top half of important features in both cases.

After further consideration, it became clear that the outcome characteristics documented as part of the data would not be initially available to make a prediction. A second training file was created to document hyperparameter tuning and training with the outcome characteristics removed from the dataset. The best hyperparameter values did not change and the accuracy of the

```
[33]: import lightgbm as lgb

[34]: #using best hyperparameters from hpo output
model = lgb.LGBMClassifier(subsample = 1.0, reg_lambda= 1, num_leaves = 180, min_child_weight = 0.1, learn

[35]: model.fit(x_train, y_train, eval_set=[(x_val,y_val)], (x_train,y_train)], verbose=20, eval_metric='logloss'

/opt/conda/lib/python3.7/site-packages/lightgbm/sklearn.py:736: UserWarning: 'verbose' argument is deprecate
d and will be removed in a future release of LightGBM. Pass 'log_evaluation()' callback via 'callbacks'
argument instead.
  log_warning("'verbose' argument is deprecated and will be removed in a future release of LightGBM. ")
[20] training's binary_logloss: 0.470722    valid_0's binary_logloss: 0.48421
[40] training's binary_logloss: 0.44058    valid_0's binary_logloss: 0.46127
[60] training's binary_logloss: 0.412532   valid_0's binary_logloss: 0.438524
[80] training's binary_logloss: 0.406808   valid_0's binary_logloss: 0.435382
[100] training's binary_logloss: 0.389867   valid_0's binary_logloss: 0.422239

[35]: LGBMClassifier(boosting_type='dart', class_weight=None, colsample_bytree=0.5,
importance_type='split', learning_rate=0.1, max_depth=-1,
min_child_samples=20, min_child_weight=0.1, min_split_gain=0.0,
n_estimators=100, n_jobs=-1, num_leaves=180, objective=None,
random_state=42, reg_alpha=0.0, reg_lambda=1, silent='warn',
subsample=1.0, subsample_for_bin=200000, subsample_freq=0)

[36]: print('Training accuracy {:.4f}'.format(model.score(x_train,y_train)))
print('Testing accuracy {:.4f}'.format(model.score(x_val,y_val)))

Training accuracy 0.8416
Testing accuracy 0.8136
```

model was slightly (about 3%) lower than when outcome characteristics were in the dataset.

Both of these models are predicting adoption versus not adopted within the specified time period that the animals are permitted to live in animal shelter facilities. The initial model includes outcome characteristics, such as timing, which may be useful for predicting adoption day of reporting, while the second model includes animal and intake characteristics that may be useful for an initial prediction when animals are first taken in and long-term planning. Productionizing the model to make prediction for operationalization for different applications became the next goal.

5. Model Productionization

After continued research about the business requirements and research goals, the outcome of Productionization attempts included two levels of sophistication:

- Low fidelity Productionization: Configure single-standing aws notebook that can be run on chron schedule at a regular cadence or be event-driven and be triggered whenever a specified event has occurred. Minimal effort and least amount of model conversion.
- High fidelity Productionization: Deploy model to endpoint. Ensuring model deployment, conversion of model output into a usable format for other applications.

For the low fidelity productionized model, the training model notebooks from AWS Sagemaker and Google co-lab were combined and compared to generate a stand alone notebook that may be run on a schedule. There is a manual section in the notebook where a subset of data may be used to make a prediction from the final model and used as needed.

For the high fidelity productionized model, the [AWS official documentation](#) on using lightGBM was studied and used as a reference throughout development. Initially, the SciKit Learn lightGBM pre-built model was used during model development. In order to simplify the process of deploying the model to an endpoint, the model was translated to use the Sagemaker pre-built lightgbm model. In the official documentation, there is sample code that was referenced to transfer best hyperparameters and deploy the model to an endpoint. The intention of having this level of productionization and sample code is to allow for the model to be integrated with systems or applications that already exist that can pull predictions from this endpoint.

Screenshots of supporting information for both productionized examples can be found in Appendix C of this report. Further, the notebooks are included in the submission, with the low fidelity Productionization code in the 'capstone_model_productionization_to_scheduled_notbook.ipynb' file and the high fidelity Productionization code in the 'capstone_model_productionization_to_endpoint.ipynb' file.

Summary

Conclusions for the project results and implementation are that the model prediction may be used to improve animal shelter operations and provides insight into the data documented for the Open Data Initiative. Model Productionization was left generic as a low fidelity notebook that may be run on a regular cadence or a deployable endpoint for integration with other applications. Overall, the model predictions were an improvement from the benchmark model and four new, original, actionable takeaway ideas were generated from data exploration and model development!

Appendix:

A. EDA Dataset Summary

Raw, uncleaned data:

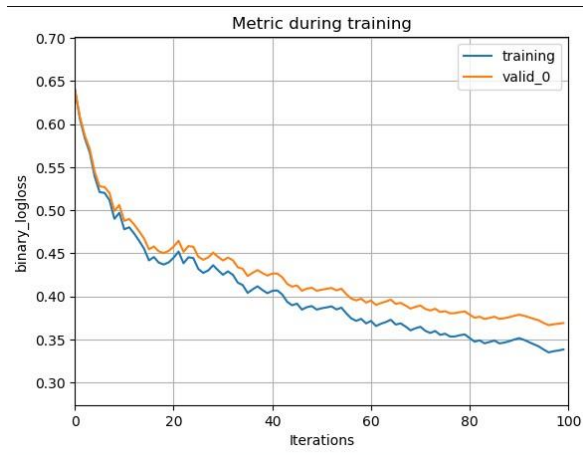
Overview	Alerts 97	Reproduction
Dataset statistics		Variable types
Number of variables	41	Categorical 26
Number of observations	79672	Numeric 15
Missing cells	43336	
Missing cells (%)	1.3%	
Duplicate rows	11	
Duplicate rows (%)	< 0.1%	
Total size in memory	24.9 MiB	
Average record size in memory	328.0 B	

Cleaned data:

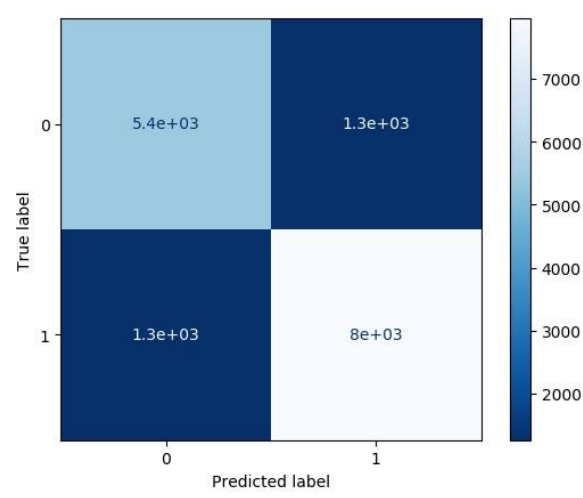
Overview	Alerts 7	Reproduction
Dataset statistics		Variable types
Number of variables	35	Categorical 18
Number of observations	79672	Numeric 15
Missing cells	12	DateTime 2
Missing cells (%)	< 0.1%	
Total size in memory	16.7 MiB	
Average record size in memory	220.3 B	

B. Machine Learning Model Training and Testing

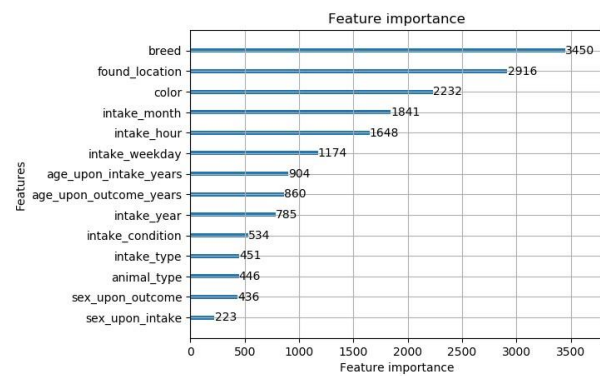
Binary log loss during training:



Model confusion matrix:



Model feature importance without outcome characteristics input to model:



C. Samples for Model Deployment

Low fidelity Productionization:

```
[ ]: #use the model to predict for a file or specific animal  
[ ]: x_val = pd.read_csv('') #insert new subset of data or specific animal record as 'csv' filetype  
[ ]: y_pred = model.predict(x_val)  
[ ]: print(y_pred)
```

High fidelity Productionization:

Sample Data per built-in LightGBM algorithm in AWS Sagemaker for Model Deployment

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
1			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2		0	1	2	10	12	2017	4	0	2	1982	527	27351	3	3	2	10	12	2017	4	14
3		1	1	2	7	12	2014	2	16	2	1982	527	23499	3	2	2	7	12	2014	0	10
4		2	1	2	6	3	2014	2	17	2	1982	527	23500	3	2	2	6	3	2014	0	14
5		3	1	2	10	4	2014	1	15	2	830	449	25821	3	1	2	10	4	2014	6	15
6		4	1	2	16	11	2013	2	11	2	1880	183	3300	2	2	2	16	11	2013	2	9
7		5	1	3	15	11	2013	3	11	2	1337	59	25821	0	3	3	15	11	2013	2	14
8		6	1	0	15	11	2014	0	19	2	1488	47	30182	3	3	0	15	11	2014	0	15
9		7	1	2	15	9	2014	1	16	2	1369	183	29229	3	3	2	15	9	2014	1	11
10		8	1	2	15	3	2014	3	15	2	1881	468	25821	3	2	2	15	3	2014	1	9
11		9	1	3	18	9	2015	0	19	2	1881	183	27171	3	3	3	18	9	2015	0	17
12		10	1	0	16	11	2015	5	13	1	941	59	8859	3	3	0	16	11	2015	0	15
13		11	1	3	14	12	2014	1	15	1	77	156	15371	3	3	3	14	12	2014	1	12
14		12	0	2	16	11	2013	0	9	2	847	56	25821	3	1	2	16	10	2013	4	15
15		13	1	2	14	2	2015	5	16	2	380	59	25821	7	1	2	14	2	2015	5	13
16		14	0	2	17	10	2016	0	12	2	669	407	14856	0	2	2	17	9	2016	6	12
17		15	1	0	13	12	2013	0	14	2	1079	407	6168	3	3	0	13	11	2013	1	17
18		16	1	3	19	12	2013	2	13	1	951	8	25821	0	0	3	19	12	2013	2	11
19		17	1	2	13	8	2014	1	16	2	1527	8	27300	3	3	2	13	8	2014	1	13
20		18	1	2	15	12	2013	1	17	2	1740	267	25821	7	1	2	15	12	2013	1	15
21		19	1	2	15	12	2013	1	17	2	1740	267	25821	7	1	2	15	12	2013	1	15
22		20	1	3	16	4	2015	6	13	2	1997	183	21381	2	3	3	16	4	2015	5	15
23		21	1	2	12	10	2013	1	12	2	1280	183	34271	3	3	2	12	10	2013	3	11
24		22	0	2	15	10	2015	4	15	2	1293	382	36187	3	3	2	15	10	2015	5	12
25		23	1	3	12	11	2013	2	17	2	617	78	30849	3	3	3	12	11	2013	2	17
26		24	1	1	13	3	2015	0	17	2	1760	69	25821	0	0	1	13	3	2015	0	15
27		25	1	2	13	9	2014	2	18	2	1850	161	3870	0	3	2	13	9	2014	2	14
28		26	1	2	12	12	2013	0	17	1	952	59	5604	3	3	2	12	12	2013	0	12
29		27	1	2	13	7	2014	6	12	2	1954	58	36121	3	3	2	12	7	2014	2	7
30		28	1	2	15	6	2016	0	12	2	1788	293	23126	3	3	2	15	6	2016	4	16
31		29	1	2	13	1	2014	3	11	1	951	32	16419	7	3	2	13	12	2013	5	16
32		30	1	2	15	5	2015	1	11	2	1280	405	21421	3	3	2	15	5	2015	6	22
33		31	1	2	14	7	2014	0	15	2	1280	405	29203	3	3	2	14	7	2014	4	17
34		32	0	3	15	11	2016	5	10	2	1050	39	28139	3	3	3	15	8	2016	6	14
35		33	1	3	14	3	2016	6	13	2	379	449	35319	0	0	3	14	3	2016	6	11
36		34	1	3	13	6	2014	4	16	3	1086	330	34236	3	3	3	13	6	2014	6	16

Deployed Endpoint

Amazon SageMaker Endpoints

Endpoints

Search endpoints

Name	ARN	Creation time	Status	Last updated
capstone-example-lightgbm-classification-2023-04-02-21-34-58-569	arn:aws:sagemaker:us-east-1:454518744659:endpoint/capstone-example-lightgbm-classification-2023-04-02-21-34-58-569	4/2/2023, 4:35:05 PM	InService	4/2/2023, 4:37:56 PM

D. Benchmark Model Comparative Analysis

The model used to benchmark with had noticeable differences and provided insight during model development. Key differences and similarities in data exploration and model development included:

- Categorization of target values for 'outcome_type': the benchmark model used the imbalanced dataset without re-segmenting or re-categorizing the outcomes of the animals. To balance the data and reframe the problem as binary classification, the definitions of the outcomes were studied and re-categorized into the two categories 'adopted' versus 'not adopted'.
- Sophistication of data cleaning: the benchmark model data was cleaned with intention to make data more uniform and standardize the input. Instead of spending time cleaning data extensively, the data was encoded, and columns dropped in inconsistencies were present.
- Model selection: In the benchmark model, decision tree-based algorithms, Decision Tree Classifier and Random Forest Classifier, were trained and used mostly default values for the hyperparameters. In the model for this project, model selection was made using AutoML technique and was down-selected based on metric performance. The light GBM algorithm was selected and then the hyperparameters tuned using the SciKit learn package. Then, then transitions to Productionization of model to AWS code was made with intention of deployment or integration with operation systems.

The intention of using a benchmark model from the 'kaggle' platform was to compare results with a ML practitioner of similar background to model produced for this project. With the final results being the highest F1 score attained was at 0.66 with a random forest model. While in this project, the highest F1 score attained was at 0.85. The percentage improvement, or percentage difference, is about 25% compared to the benchmark model. This may be considered a measure of improvement from using AutoML and hyperparameter tuning techniques during model development.

A notable difference between the models was the selected features used for model training. It seems the benchmark model excluded the time features and only used the animal characteristics to attempt to predict the outcome. While for the model in this project, time features were included for intake and outcomes. It should also be noted, on the initial set out to complete this project, one of the goals was to observe and correct for seasonality in the data. After further reading and understanding of seasonality in time-series forecasting and modeling, the seasonality aspirations were dropped as the problem was reframed as binary classification problem.

Comparing approaches and outcomes with the benchmark models, a new perspective is gained. The location and time data ended up playing a significant role based on feature importance plots generated after model training. It can be noted that the benchmark model plotted feature importance after training after dropping columns, as if an afterthought in the process. Feature importance was seen as a tool to be used to select columns to drop, and this was a difference in the approaches. Lastly, as model development and building occurred, it became evident that model selection and hyperparameter tuning were not a part of the benchmark model process. Having been trained as a ML practitioner some of the trending tools that assist with this process, the model selection and hyperparameter tuning was less time intensive then the actual model training and Productionization. Not only was the project model better balance in precision and recall, but this part of the process helped justify the selection of the tree-based models for this problem with a bit more justification and data from the AutoML outputs.

E. Model Operationalization Possibilities: Key Takeaways

The model developed in this project was developed and trained to predict the outcome of animal adoption within allowed permitted time in the animal shelter. Applications of this type of technology and outcome include the following actionable solution ideas:

- Operational Guide Upon Intake
 - Upon initial intake, outcome characteristics would be excluded from the data, however, using the animal and intake characteristics, an initial adoption prediction could be made. With this initial prediction, there can be operational action plans for animals anticipated to be adopted versus not adopted. This may include transferring not adopted animals sooner than planned or targeting marketing campaigns for animals not likely to be adopted within the allowed time frame.
- Animal Shelter Personnel Education and Training
 - Informing animal shelter personnel of high adoption rates for certain animal characteristics may assist personnel in planning for animal care and understanding of adoption rates. This understanding may help with operational strategies in communications to the public and funding sources.
- Animal Relocation Network
 - A bit more ambitious, but if more models are trained across a network or animal shelters, having the probability of adoption of animals at different locations available in this network, and animals transitioned to the location they would most likely be adopted in could be solution to reducing non-adopted animals.
- Animal Name Generator Application
 - As more and more data is collected, another potential solution could be an animal name generator that used the prediction data to adjust the animals name. The application could optimize or maximize the probability of being adopted by leveraging name data and the historical data of animals taking into animal shelters with names of adopted animals.

These are just some examples of the ideas that were generated from studying the data and the model development. People a bit more involved with the domain and operations may have more ideas about how this adoption prediction may be useful.

Throughout the development of the model and brainstorming of the operationalization possibilities there were major takeaways and lessons learned about the usefulness of data in improving machine learning model development operations and data analysis:

- Without analyzing the data initially, there was some misconceptions about what may be found in the data; after studying and cleaning the data, it became apparent the outcomes were complex and, at times, unanticipated because the data is a result of human decision-making.
- Data is contextual; you can have large datasets and types of data, however, the context and business requirements play a crucial role in understanding what is noise and what is useful. ML model development tools can also help in filtering most types of data.
- Core concepts of model Productionization remain unchanged for various business or operational contexts. The core of Productionization is transition models to AWS components and endpoints.

F. References

<https://learn.udacity.com/nanodegrees/nd189/parts/cd0549/lessons/a04d5089-4e2d-42e4-a6e3-1f6d3ee055c6/concepts/a04d5089-4e2d-42e4-a6e3-1f6d3ee055c6-submit-project>

<https://www.kaggle.com/code/wenlie/exploring-and-predicting-the-animal-s-outcomes>

[https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)

[learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split)

Udacity course: Dataset Principles, Lesson: Model Deployment Workflow

[https://sagemaker-](https://sagemaker-examples.readthedocs.io/en/latest/aws_sagemaker_studio/getting_started/xgboost_customer_churn_studio.html)

[examples.readthedocs.io/en/latest/aws_sagemaker_studio/getting_started/xgboost_customer_churn_studio.html](https://sagemaker-examples.readthedocs.io/en/latest/aws_sagemaker_studio/getting_started/xgboost_customer_churn_studio.html)

<https://github.com/autogluon/autogluon/issues/274>

<https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html>

[https://github.com/aws/amazon-sagemaker-](https://github.com/aws/amazon-sagemaker-examples/blob/main/introduction_to_amazon_algorithms/lightgbm_catboost_tabular/Amazon_Tabular_Classification_LightGBM_CatBoost.ipynb)

[examples/blob/main/introduction_to_amazon_algorithms/lightgbm_catboost_tabular/Amazon_Tabular_Classification_LightGBM_CatBoost.ipynb](https://github.com/aws/amazon-sagemaker-examples/blob/main/introduction_to_amazon_algorithms/lightgbm_catboost_tabular/Amazon_Tabular_Classification_LightGBM_CatBoost.ipynb)

<https://sparkbyexamples.com/pandas/pandas-change-position-of-a-column/>

<https://stackoverflow.com/questions/64524749/lightgbm-valueerror-series-dtypes-must-be-int-float-or-bool>

<https://aws.amazon.com/blogs/machine-learning/amazon-sagemaker-built-in-lightgbm-now-offers-distributed-training-using-dask/>

<https://www.kaggle.com/code/prashant111/lightgbm-classifier-in-python>

<https://towardsdatascience.com/how-to-beat-the-heck-out-of-xgboost-with-lightgbm-comprehensive-tutorial-5eba52195997>

<https://docs.aws.amazon.com/sagemaker/latest/dg/lightgbm-tuning.html>

<https://dataintegration.info/optimize-hyperparameters-with-amazon-sagemaker-automatic-model-tuning>

<https://www.analyticsvidhya.com/blog/2021/08/complete-guide-on-how-to-use-lightgbm-in-python/>

https://scikit-learn.org/stable/modules/grid_search.html

<https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/>

<https://github.com/NandhiniN85/AWS-SageMaker/blob/master/Training%20notebook.ipynb>

[https://github.com/aws/amazon-sagemaker-](https://github.com/aws/amazon-sagemaker-examples/blob/main/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb)

[examples/blob/main/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb](https://github.com/aws/amazon-sagemaker-examples/blob/main/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb)