Machine Learning Engineer Nanodegree

Ariel Mendez

Capstone Project

April 18, 2019

I. Definition

Project Overview

Millions of domestic animals around the world are homeless and living an unhealthy life in streets or other difficult places. Unfortunately, animal shelters and organizations can't afford to sustain a large population and many pets are euthanized[1].

PetFinder is a Malaysian organization which sustains a database of more than 150,000 animals that are in adoption. They post an online profile of the animals so people who are looking to adopt can have an accessible way to do it.

Recently PetFinder opened a competition in the data science platform Kaggle. The objective of this competition is to help PetFinder by providing an algorithm that predicts the speed in which an animal is going to be adopted based on the metadata of its profile. Arriving at a useful solution could help improve the profile of thousands of animals and give them a better chance to find a new family.

Problem Statement

According to the description in Kaggle, the problem is to "predict the speed at which a pet is adopted". In case the profile gives information about several pets, the speed is defined as the speed of adoption of all pets in the profile.

The speed of adoption is determined by a categorical variable in the range from 0 to 4. The values are defined as follows:

0 - Pet was adopted on the same day as it was listed.

1 - Pet was adopted between 1 and 7 days (1st week) after being listed.

2 - Pet was adopted between 8 and 30 days (1st month) after being listed.

3 - Pet was adopted between 31 and 90 days (2nd & 3rd month) after being listed.

4 - No adoption after 100 days of being listed.

The task is a supervised learning classification problem. For this reason, I experimented with different classifiers to find which modeled the data better. The algorithms used were:

- Random Forest
- XGBoost

[1] Pets by the numbers

- LightGBM

The final algorithm should label new observations with accurate speed category.

Metrics

PetFinder competition score submissions using quadratic weighted kappa[2]. This metric is a statistical measure of agreement between two raters (actual and predicted values) who label observations with a finite set of categorical values. Weighted kappa refers to assign a weight to the disagreement i.e. difference between actual and predicted rating scores. This implies that categories are ordered, in this case ratings from 0 to 4.

Therefore, three matrices are required to generate a score:

- Observation matrix O – N by N histogram matrix such that each cell has the number of observations that have rating **i** (actual) and received a predicted rating **j**.
- Expected matrix E – The outer product of vector of ratings i.e. the sum of each row of O and vector of predictions i.e. the sum of each column of O.
- Weight matrix W – N by N matrix where each cell has the value $(i - j)^2$ and the main diagonal has 0s.

Where N = 5 (ratings from 0 to 4) and "i", "j" refers to rows and columns.

Given these matrices weighted kappa is calculated as follows[3]:

$$kw = 1 - \frac{\sum_{i,j} w_{i,j} \cdot p_{i,j}}{\sum_{i,j} w_{i,j} \cdot e_{i,j}}$$

The score ranges from 0 to 1, where 0 indicates no agreement among the raters and 1 represent a complete agreement. Negative values are possible; however, these values imply that the agreement is worse than random[4].

This metric is appropriate for this task since the prediction scores are categorical values from 0 to 4 where order implies a slower speed of adoption.

II. Analysis

Data Exploration

The data files provided by PetFinder and used in this project are:

[2] PetFinder competition evaluation: https://www.kaggle.com/c/petfinder-adoption-prediction/overview/evaluation

[3] Weighted Cohen's Kappa explanation

[4] Cohen's kappa article

- train.csv – tabular data containing profile features and the adoption speed target variable.
- test.csv – tabular data containing profile features
- test_metadata[5] and train_metadata
- test_sentiment[6] and train_sentiment
- breed_labels –PetID and breed of pet
- color_labels – PetID and color of pet
- state_labels – PetID state location state

The pet features present in the train/test files are[7]:

- PetID - Unique hash ID of pet profile
- AdoptionSpeed - Categorical speed of adoption. Lower is faster. Predict variable
- Type - Type of animal (1 = Dog, 2 = Cat)
- Name - Name of pet (Empty if not named)
- Age - Age of pet when listed, in months
- Breed1 - Primary breed of pet (Refer to BreedLabels dictionary)
- Breed2 - Secondary breed of pet, if pet is of mixed breed (Refer to BreedLabels dictionary)
- Gender - Gender of pet (1 = Male, 2 = Female, 3 = Mixed, if profile represents group of pets)
- Color1 - Color 1 of pet (Refer to ColorLabels dictionary)
- Color2 - Color 2 of pet (Refer to ColorLabels dictionary)
- Color3 - Color 3 of pet (Refer to ColorLabels dictionary)
- MaturitySize - Size at maturity (1 = Small, 2 = Medium, 3 = Large, 4 = Extra Large, 0 = Not Specified)
- FurLength - Fur length (1 = Short, 2 = Medium, 3 = Long, 0 = Not Specified)
- Vaccinated - Pet has been vaccinated (1 = Yes, 2 = No, 3 = Not Sure)
- Dewormed - Pet has been dewormed (1 = Yes, 2 = No, 3 = Not Sure)
- Sterilized - Pet has been spayed / neutered (1 = Yes, 2 = No, 3 = Not Sure)
- Health - Health Condition (1 = Healthy, 2 = Minor Injury, 3 = Serious Injury, 0 = Not Specified)
- Quantity - Number of pets represented in profile
- Fee - Adoption fee (0 = Free)
- State - State location in Malaysia (Refer to StateLabels dictionary)
- RescuerID - Unique hash ID of rescuer
- VideoAmt - Total uploaded videos for this pet

[5] Test/train_metadata directory that contains json files obtained after running each pet image through the Google's Vision API.
[6] Test/train_sentiment directory that contains json files obtained after running each pet description through the Google's Natural Language AP.
[7] Kaggle PetFinder data description section: https://www.kaggle.com/c/petfinder-adoption-prediction/data

- PhotoAmt - Total uploaded photos for this pet
- Description - Profile write-up for this pet. The primary language used is English, with some in Malay or Chinese.

As noted above, the target variable for this project is "AdoptionSpeed".

From the features described above, the categorical variables are:

- Type, Breed 1/2, Gender, Color 1..3, MaturitySize, FurLength, Vaccinated, Dewormed, Sterilized, Health

The continues variables are:

- Age, Quantity, RescuerID, PhotoAmt, VideoAmt

Descriptive statistics from these variables are summarized in the next table.

	Age	Quantity	PhotoAmt	Fee	VideoAmt
count	14993.000000	14993.000000	14993.000000	14993.000000	14993.000000
mean	10.452078	1.576069	3.889215	21.259988	0.056760
std	18.155790	1.472477	3.487810	78.414548	0.346185
min	0.000000	1.000000	0.000000	0.000000	0.000000
25%	2.000000	1.000000	2.000000	0.000000	0.000000
50%	3.000000	1.000000	3.000000	0.000000	0.000000
75%	12.000000	1.000000	5.000000	0.000000	0.000000
max	255.000000	20.000000	30.000000	3000.000000	8.000000

Figure 1 Descriptive statistics of continuous variables

There are 1269 nan[8] values in the training data set, from which 1257 come from the name column and 12 come from the description column. The test data set has 411 nan values coming from 'Name' and 1 coming from 'Description'.

The next table shows the number of outliers from the continuous data set.

Outliers were defined as values $< Q_1 - 1.5 \cdot IQR$ or values $> Q_3 + 1.5 \cdot IQR$

Feature	Number of outliers	Percentage of data
Age	1501	10.01 %
Quatity	3428	22.86 %
PhotoAmt	922	6.15 %
VideoAmt	2330	15.54 %
Fee	574	3.82%

[8] Nan – not a number

Given that the mean value is affected by the presence of outliers, the median is preferred as a measure of central tendency[9].

Median values of the 5 continuous variables are:

Feature	Median
Age	3.0
Quatity	1.0
PhotoAmt	3.0
VideoAmt	0.0
Fee	0.0

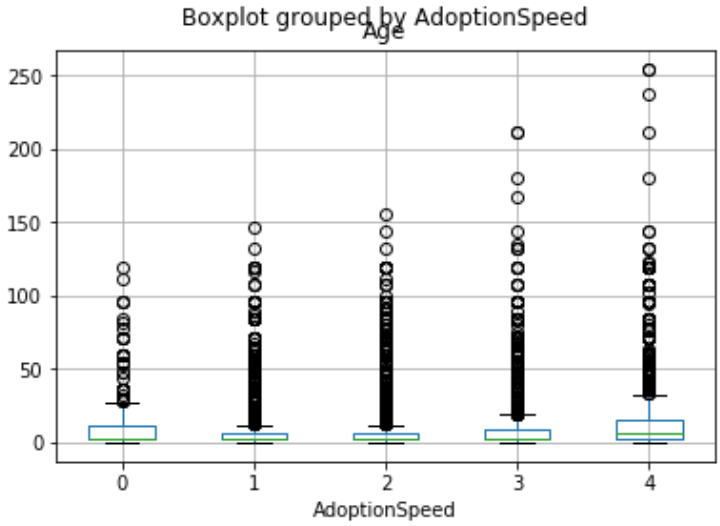
Comparing the median values to their respective mean, it is easy to observe that outliers were affecting the score of central tendency. The median age is 3 months compared to the mean of 10 months. This is a better value to get a sense of the distribution because 51.84 % of the observations have 3 or less months. Also, 84.45% of the observations have a fee of 0 which is described by the median value of 0 compared to the mean of 21.26 monetary units.

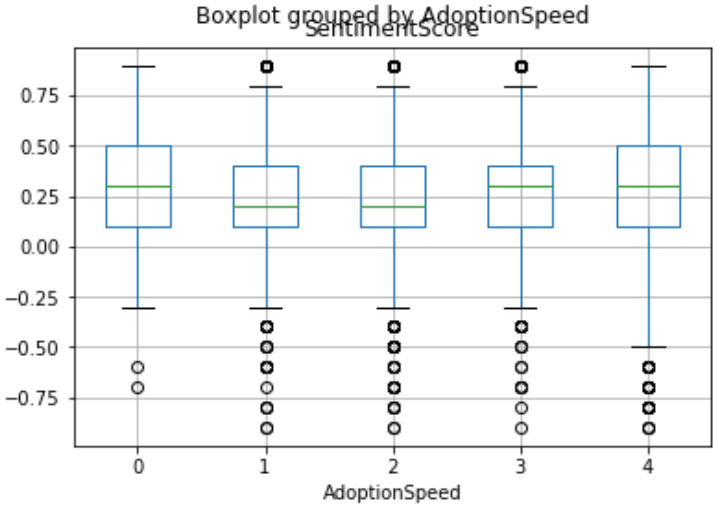
Exploratory Visualization

The next plots show the visualizations created during the data analysis step. Plots are generated from training data.

Univariate Plots

The first plots show the boxplots of Age and Sentiment Score grouped by Adoption Speed categories.



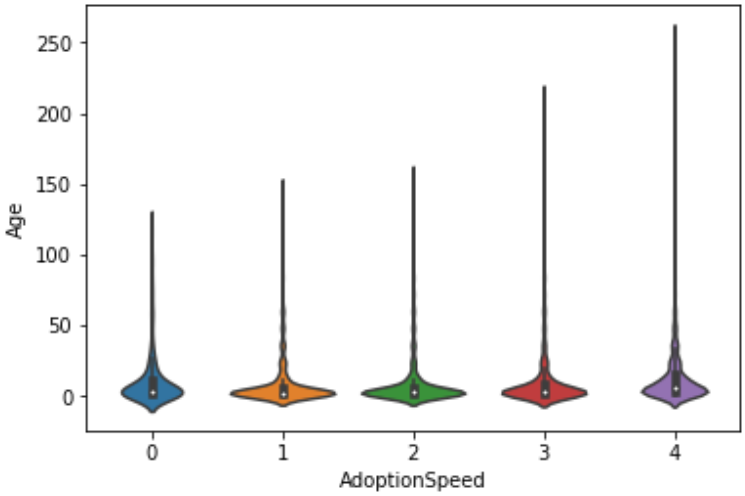


[9] Measures of Central Tendency - How do outliers influence the measures of central tendency?

The age box plot shows how most observations have less than 25 months. From this age onwards most of the observations are labeled as outliers. Calculating this value from the training set I found that 13452 or 89.7% of the observations have less than 25 months.

From the sentiment score[10] boxplot an interesting point to consider is that pets that have an Adoption Speed value 0 and a value of 4 have 50% of their observations centered around the same values. That is IQR (Interquartile Range) of both distributions is approximately 0.5 – 0.125 = 0.375 and mean of 0.3.

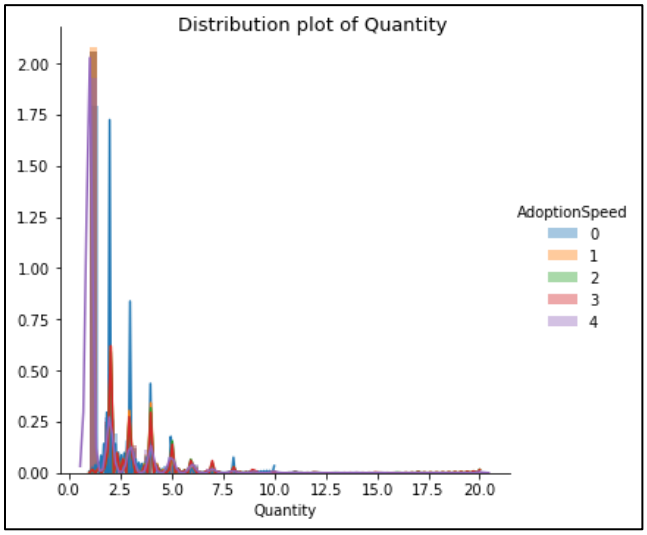
Violin plot

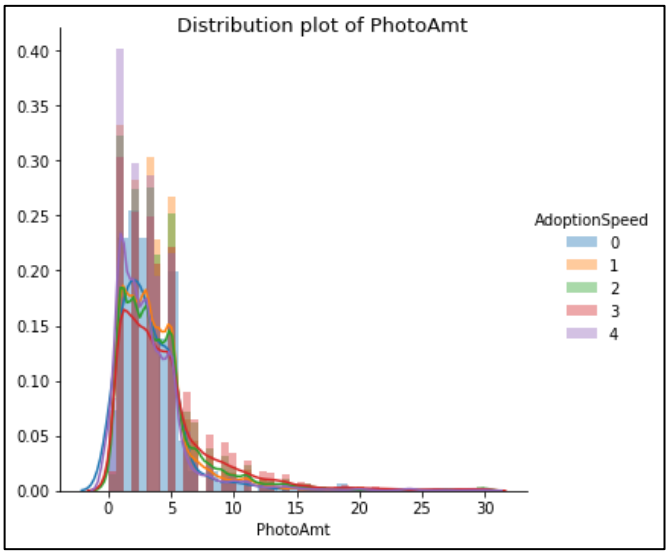


The violin plot of Age demonstrates more clearly how the outliers move farther with each adoption speed category.

This information could be used by a model, like a decision tree, to label outliers. For example, if the outlier has an age bigger than 150 months most probably it belongs to category 3 or 4, and if it has a value bigger than 200 most probably it will be labeled with a 4.

Density plots





[10] Sentiment score refers to the emotional score of the pet description. It was extracted from the sentiment metadata directory.

The distribution plot of Quantity (number of pets represented in profile) shows that profiles which contain 2 or 3 pets have a higher density of adoption speed values of 0, 1, 2 and 3. In other words, the density of pets with adoption speed 4 drops rapidly compared to the other speed values.

The distribution plot of PhotoAmt, total uploaded photos of pet, shows that the 5 distributions are spliced and have a similar shape. Finally, most pets have 1 to 5 photos.

Bar charts – Categorical plots

Bar charts were used to visualize categorical variables, for the ease to observe the number of observations in each category.

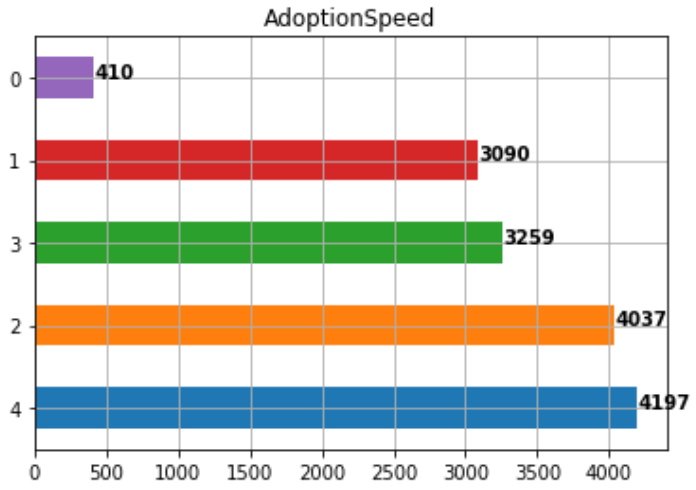


Figure 2 Categorical speed of adoption

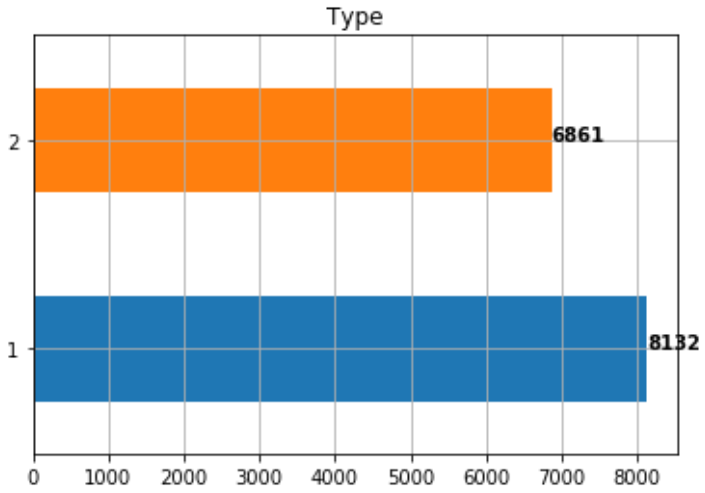


Figure 3Type of animal (1 = Dog, 2 = Cat)

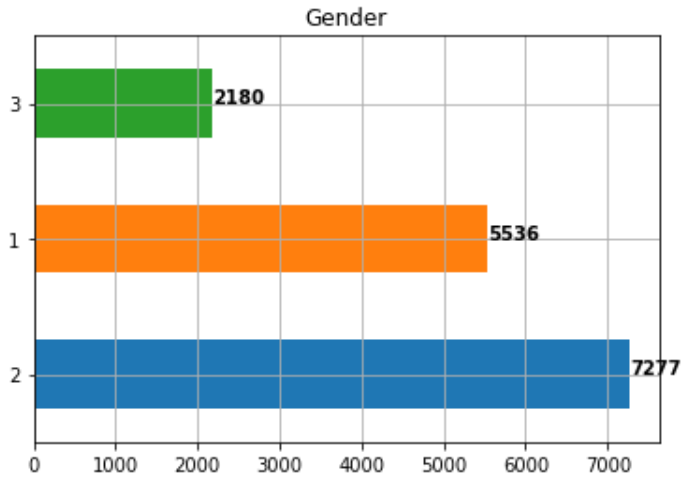


Figure 5 Gender of pet (1 = Male, 2 = Female, 3 = Mixed)

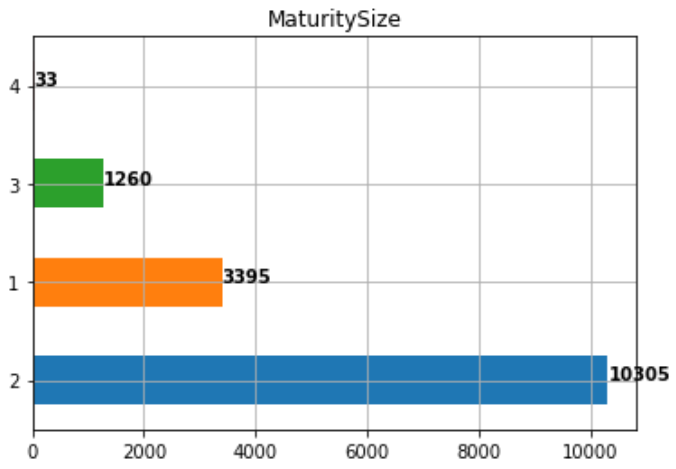


Figure 4 Size at maturity (1 = Small, 2 = Medium, 3 = Large, 4 = Extra Large, 0 = Not Specified)

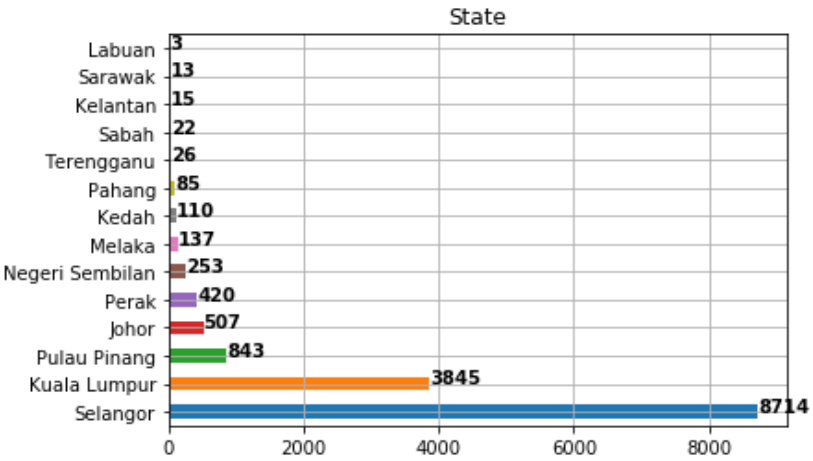


Figure 6 Number of pets per State

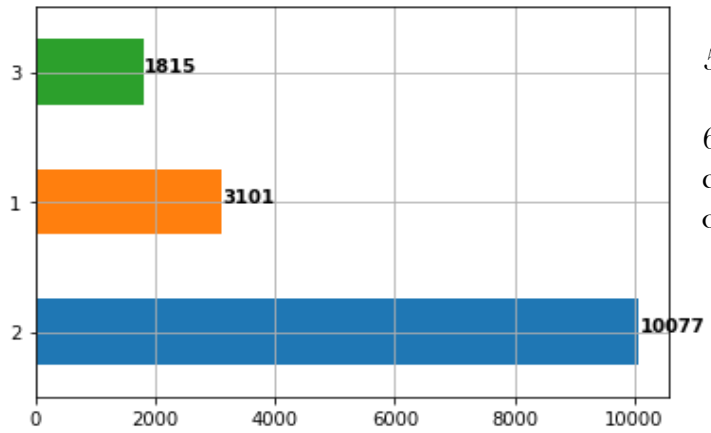


Figure 7 Pet has been spayed / neutered (1 = Yes, 2 = No, 3 = Not Sure)

Summary of observations of bar graphs:

Unfortunately, 27.9% of pets were not adopted after 100 days of being listed in PetFinder.

There are 1271 more dogs than cats.

There are 1741 more female observations than males.

68.7% of pets are of medium size and 22.6% are of small size.

58.1% of pets are in Selangor.

67.2% of pets are sterilized, this fact could be correlated with the high number of female observations.

Bi-variate visualizations - Pair plot

The next plot shows the relationship between quantitative variables.

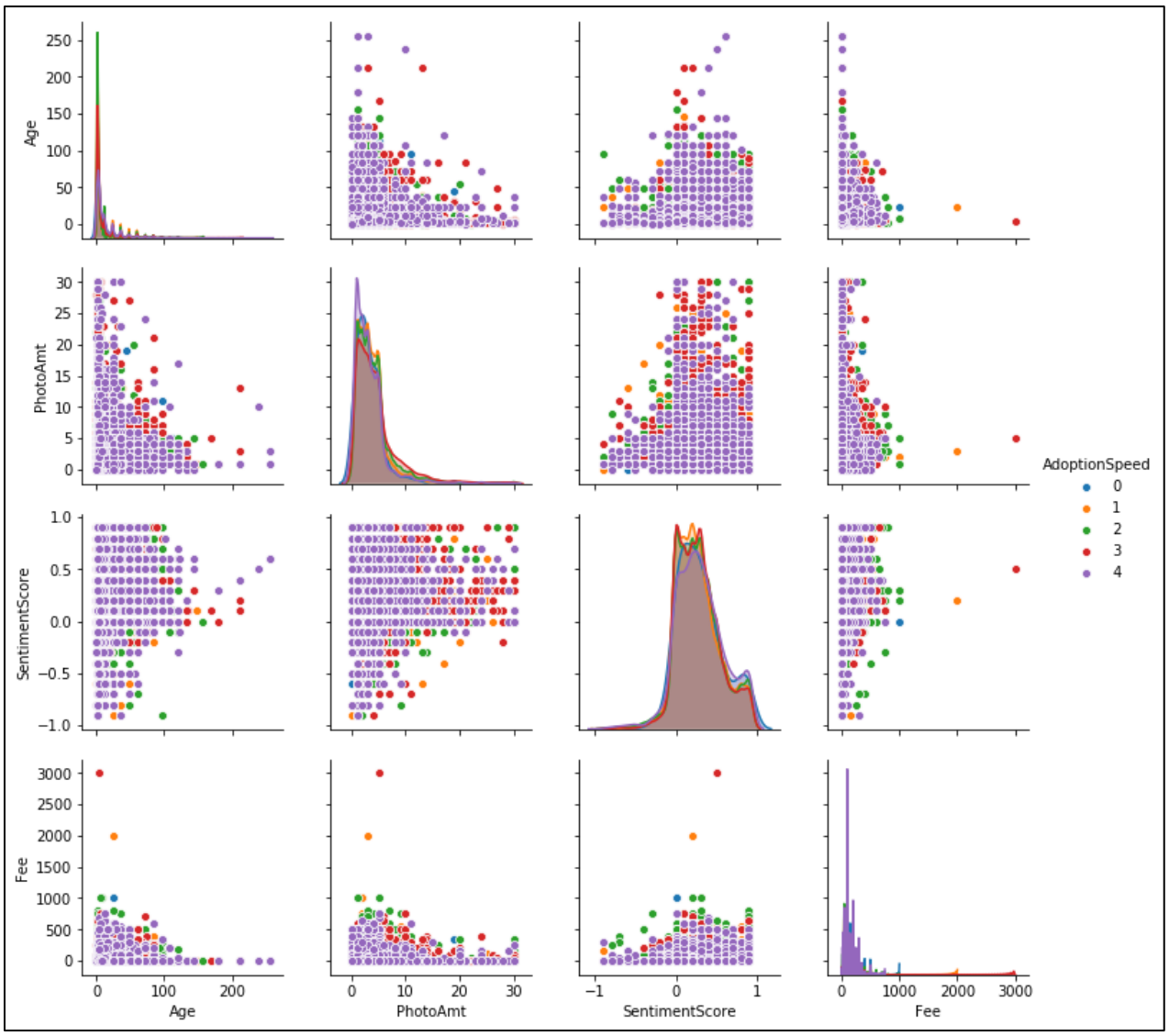


Figure 8 Pair plot of continuous variables

The scatterplot of Fee vs Age shows how the majority pets that have a fee for adoption are the ones who have less than 100 months. Age is also related to the number of photos of pets, pets that get older get less photos registered.

An interesting observation is in the PhotoAmt vs SentimentScore plot. It seems that above 15 photos and sentiment bigger than 0 the number of purple observations decreases, and it can be seen more presence of red observations. These kinds of data separations can be exploited by models based on decision trees.

Algorithms and Techniques

The task is a classification problem that requires training a model to predict qualitative labels. For this reason, I trained different classifiers on the training data to find which models the data better.

XGBoost

Extreme Gradient Boosting library provides an implementation of gradient boosting machines. XGBoost algorithm uses decision tree ensembles which consist of classification and regression trees (CART).

XGBoost implementation requires all training data to be in numerical form. As noted, before, just the name, description and IDs are not numbers. I considered the description feature to be important, so I transformed it into a numerical feature by applying a preprocessing step.

XGBoost model has 3 main types of parameters[11]:

1. General Parameters
    a. Booster – type of model to run at each iteration
    b. Silent – boolean mode to print messages of running tasks
    c. Nthread – number of threads for parallel processing
2. Booster Parameters (for tree model)
    a. Eta – learning rate, typical values range from 0.01 – 0.2
    b. Min_child_weight – minimum number of instances needed to be in each node
    c. Max depth and leaf nodes – maximum depth of tree and leaves in a tree.
    d. Gamma – minimum loss reduction required to make a further partition
    e. Lambda – L2 regularization term
    f. Alpha – L1 regularization term
3. Learning Task Parameters
    a. Objective – loss function to be minimized
    b. Eval_metric – metric to be used for validation data

The ensemble tree model is trained using gradient boosting technique, which uses an additive strategy to iteratively fix and improve the predictions of trees.

LightGBM

LightGBM is a boosting framework that uses tree models for learning. The peculiarity of this algorithm is that it grows vertically instead of horizontally like other tree-based algorithms. The leaf that will be divided is the one that has the max delta loss. Growing leaf-wise can reduce more loss than level wise[12].

[11] XGBoost parameter tuning article

[12] What is LightGBM:

Some important parameters[13] are:

1. Core
    a. Task – train or prediction
    b. Application – regression or classification task
    c. Learning rate – magnitude of change in estimates of each tree on final outcome
    d. Num_leaves – number of leaves
    e. Metric – loss function
2. Control parameters
    a. Max_depth: maxim depth of tree
    b. Lambda – regularization values
    c. Min_data_in_leaf – minimum number of records a leaf may have.
3. IO
    a. Max_bin – max number of bins that feature values will be bucketed in.

As the training of LightGBM is leaf-wise it is more prone to overfitting if there is no sufficient data. In this case, the training data set has 14993 observations which is a reasonable number of points that can be model by this algorithm.

Random Forest

This algorithm is an ensemble learning method based on decision trees. It is trained by using features aggregating which differs from bootstrap aggregating by selecting at each candidate split a random subset of features.

The final classification is obtained by averaging the predictions of all trees.

Important parameters of sklearn random forest implementation are[14]:

a) N_estimators – number of trees in forest.
b) Max_features – number of features to consider when doing the best split.
c) Min_samples_leaf – minimum number of training samples at left and right branch to be at a leaf node.
d) Max_depth - max number of levels in each tree
e) Bootstrap – if true a random sample is used to train trees, otherwise the whole data set is used.

The score of random forest could be used as a baseline model to improve and compare with XGBoost and Light GBM scores.

[13] LightGBM parameters: https://lightgbm.readthedocs.io/en/latest/Parameters.html
[14] Random forest parameters: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Benchmark

I used as an initial benchmark the classifications generated by the random forest algorithm without hyperparameter tuning. Along with this, the kaggle leaderboard is useful to compare my score to the scores of other competitors.

Kaggle platform allows competitors to explain their approach to the problem using Kernels. These Kernels are Jupyter notebooks with the explanation of the techniques used by the author. The competition leaderboard displays the title of public Kernels alongside the score of the author. This also works as a benchmark because kernel titles are often techniques that alongside the score help me compare the results of similar techniques.

The initial random forest model used the next parameters:

- n_estimators – 1750
- min_samples_leaf – 3
- max_features – sqrt
- max_depth - 70

The output of this model is placed into a csv and uploaded to Kaggle. This submission file is scored using the quadratic weighted kappa. It obtained a score of 0.19032 which is better than the scores of competitors that used Trees (score = 0.15193) or logistic regression (score = 0.14621).

III. Methodology

Data Preprocessing

The train sentiment directory contains metadata related to the description of the pet after being processed by Googles Natural Language API. It comes in json format and it has several attributes related to the sentiment of each sentence and the overall document sentiment. I decided to extract the document sentiment score which is a value between -1 and 1. This value corresponds to the overall emotion of the document[15].

To accomplish this, I created a sentiment score dictionary which maps PetIDs with a sentiment score. I applied this preprocessing step with the training and testing sets. After generating the sentiment dictionaries, I mapped the train and test data frames with the sentiment dictionary using the ped id as keys. This step generated a new sentiment score feature in each data frame. Pets which do not have a sentiment score associated, received a value of 0.

As mentioned in the algorithms section, all features should be numerical if I wanted to use XGBoost or LightGBM libraries. For this reason, I applied a TFIDF transformation to the description feature by using TfidfVectorizer class from scikit learn library[16]. Before applying this

[15] Google Natural Language API: https://cloud.google.com/natural-language/docs/basics

[16] TfidfVectorizer documentation from scikit learn.

transformation, all nan values were replaced with an empty string. The output matrix of TF-IDF features had 14993 rows and 20914 columns. To incorporate this information to the training data a reduction in the dimensions of the matrix was required.

For applied dimensionality reduction using truncated SVD. Scikit learn provides the object TruncatedSVD which according to the documentation works on term count/tf-idf matrices. This process is also known as latent sematic analysis (LSA). I set to 1 the number of components. In other words, the TF-IDF matrix is going to be summarized by one component[17]. I followed the same steps for the description feature in the testing set.

Columns, Name, RescuerID, and PetID were dropped from the data set. Finally, the continuous variables were normalized by applying min/max normalization and standardized subtracting the mean and divide by the standard deviation of each feature.

After the data preprocessing steps described above, the training data set contains 22 numerical columns and the testing set has 21 columns because of not having the target column.

Implementation

I started the implementation by working with the random forest algorithm. To implement the benchmark model defined earlier, I imported RandomizedSearchCV from scikit learn. The purpose was to test a random set of parameter values and from the best estimator generated, take its parameters and define a new set of parameters to be tested by a grid search strategy.

The original random grid was defined as follows:

- 'n_estimators': [100, 250, 500, 750, 1000, 1500, 1750, 2000]
- 'max_features': ['auto', 'sqrt', 'log2']
- 'min_samples_leaf': [1,2,3,4,5,6]
- 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]

The best parameters found were the ones reported in the benchmark section. The difference between random search and grid search is that the second one will test all the combination of values while the first one is going to select just a sample.

The new refined parameter list of values was:

- 'n_estimators': [1500, 1750, 2000, 2250, 2500]
- 'min_samples_leaf': [3, 4, 5]
- 'max_depth': [60,70, 80, 90, 100]

Alongside with this parameter dictionary the GridSearch object required some parameters like cv (number of folds in StratifiedKFold) which was set to 2 and n_jobs (number of parallel jobs) which was set to -1 to use all available processors.

[17] I learned the application of TFIDF and dimensionality reduction from Kernel "Single XGBoost model" and the answer to the question "How to reduce dimension for text document dataset?" in StackExchange.

Running this process took 56 minutes to complete. The submission generated by the best estimator found in grid search obtained a score of 0.21771 which was an improvement from the benchmark score of 0.19032.

For the implementation of XGBoost I followed the next process:

1. Split the training data by using a stratified K fold strategy with 10 folds. Each data fold preserves the proportion of observations from each feature. This strategy allowed me to have a validation set for hyper-parameter tuning.
2. For each fold I trained the algorithm using the default parameters present in the implementation.
3. After each training step I made a prediction with the model generated and saved the results into a matrix of shape (number of observations, number of folds).
4. The final output was obtained by averaging each row of the matrix of scores. This output is formatted into a csv file and uploaded to kaggle for evaluation.
5. The score obtained by XGBoost using the default parameters was 0.19909 which is not an improvement over the final score obtained by random forest.

LightGBM implementation followed the same steps used for XGBoost.

- Split the data with StratifiedKFold with 10 folds
- For each fold train the algorithm and generate a prediction.
- Average each prediction to generate the result.
- Upload the csv file for evaluation

For both models the validation set obtained by the splitting the data was used to observe the scores obtained in training and validation sets during training.

Training the algorithm with the default parameters and averaging the 10 predictions gave a score of 0.23165 which is better than the score obtained by the default implementation of XGBoost.

Refinement

The process of refinement of the algorithms consisted in tune the hyperparameters to better performance.

The initial score generated in the implementation of XGBoost was 0.19909. To tune the default parameters of XGBoost I defined the next parameter values dictionary[18]:

- eval_metric: 'merror' – It is used for multiclass classification error rate
- 'eta': 0.01, - Step size shrinkage
- 'max_depth': 3,
- 'subsample': 0.8,
- 'colsample_bytree': 1,

[18] Followed some of the advices given in: Fine-tuning XGBoost and Complete Guide to Parameter Tuning in XGBoost

- 'gamma': 0,
- 'objective': 'multi:softmax', - softmax objective which returns predicted class
- 'num_class': 5 – number of classes.

Alongside these values, the training function num_boost_round was set to 200 and in the prediction function tree limit was set to the best tree limit found in training.

The score obtained by redefining the parameters was 0.27234, which is an improvement from both the random forest model and the initial XGBoost model.

Max_depth	Score
3	0.27234,
4	0.28799
5	0.29304
6	0.29392
7	0.30241
8	0.30269
9	0.30146

The best score is obtained with max_depth of 8.

- Changing the eta to 0.05 improved the score to 0.31366. An eta value of 0.04 dropped to 0.3069 and a value of 0.06 dropped to 0.31076. Hence, eta was set to 0.05.
- Subsample was set to 0.5 which improved the score to 0.31961
- Trying different values for colsample_bytree gave the next results:

colsample_bytree	Score
0.4	0.32106
0.6	0.32257
0.8	0.32103

Finally setting gamma to 2 improved to score to 0.32601.

For LightGBM I redefine the default parameters to the next values:

- 'min_data_in_leaf': 100,
- 'application': 'multiclass',
- 'num_class': 5,
- 'metric': 'softmax',
- 'learning_rate': 0.1,
- 'num_leaves': 100

The predict function parameter num_iteration was set to the best iteration found during training. This limits the number of iterations in prediction[19]. The score of this model was 0.31521 which is an improvement over 0.23165 the score obtained with default parameters.

[19] Data Structure API: https://lightgbm.readthedocs.io/en/latest/Python-API.html

Testing different values of min_data_in_leaf I found that the best number was 300.

min_data_in_leaf	Score
200	0.31904
300	0.32189
400	0.31837

Changing the learning rate and num_leaves decreased the score, so these values remained the same. On the other hand, the next values were found by testing different values of max_bin

max_bin	Score
200	0.32278
300	0.32497
400	0.32045

The final score of LighttGBM used the next parameters:

- 'eval_metric': 'merror',
- 'eta': 0.05,
- 'max_depth': 8,
- 'subsample': 0.5,
- 'colsample_bytree': 0.8,
- 'gamma': 2,
- 'objective': 'multi:softmax',
- 'num_class': 5

The score was **0.32497** which was a little lower than the score obtained by XGBoost.

IV. Results

Model Evaluation and Validation

I choose XGBoost as the final model with a final score of 0.32601. After hyper-parameter tuning it improved a little more than LightGBM. The final model was trained with the next parameters:

1. Booster parameters
    a. 'eta': 0.05,
    b. 'max_depth': 8,
    c. 'gamma': 2,
    d. 'subsample': 0.5,
    e. 'colsample_bytree': 0.8,
2. Learning task parameters
    a. 'eval_metric': 'merror',
        i. Selected merror because it is used in multiclass classification problems.

b. 'objective': 'multi:softmax',
c. 'num_class': 5

The train process used 200 boost rounds. The implementation can be inspected in the complementary jupyter notebook.

I also observed that predictions were more accurate after averaging the 10 predictions generated by the training/prediction process in each fold.

The train_XGBoost function (in jupyter notebook) returns the model and the output matrix with the predictions of each fold. The predictions on the testing data using the model returned obtained a score of 0.30754 which is lower than the one obtained by averaging.

The next table shows the importance of each feature by displaying the average gain across all splits.

'Breed1': 3.249	'Age': 3.467	'Vaccinated': 2.639
'FurLength': 2.598	'SentimentScore': 2.449	'Sterilized': 4.131
'TFIDF_Description_0': 2.568	'Color3': 2.357	'MaturitySize': 2.476
'Color2': 2.389	'Gender': 2.588	'State': 2.576
'PhotoAmt': 2.906	'Dewormed': 2.545	'VideoAmt': 2.040
'Breed2': 2.598	'Color1': 2.390	'Health': 2.225
'Quantity': 2.827	'Fee': 2.426	'Type': 2.783

As it can be seen Age and Sterilized are the most important features, for this reason to check how robust the model is I added one standard deviation from age feature to all values of Age from the testing set.

The next table shows the frequency of Adoption Speed labels on the prediction set before and after altering the data.

Adoption Speed	Count of values original test set	Count of values altered test set
4	1270	2812
3	1131	570
2	986	311
1	584	278
0	1	1

1542 observations were now labeled with 4, however, the only value that was labeled with 0 stayed with 0. This is an indicator that the model generalized well by maintaining the importance of features.

Justification

The final model predictions can be compared with the target variable present in the train data. The train data has 14993 observations while the test data frame has 3972 observations, however the proportion of values is similar in both data frames.

Adoption Speed	Proportion of Adoption Speed values in train data	Proportion of Adoption Speed values in prediction
4	0.279931	0.319738
3	0.269259	0.284743
2	0.217368	0.248238
1	0.206096	0.147029
0	0.027346	0.000252

All labels follow the same pattern, label 0 still has less than 3% percent while labels 2, 3 and 4 have the more than 20% of observations each. Maintaining the similar proportions works as a signal of the capacity of the final model to catch feature patterns.

The next table shows the improvement achieved over the benchmark score:

Benchmark (Random Forest)	Base XGBoost model	Final XGBoost model
0.19032	0.19909	**0.32601**

V. Conclusion

Free-Form Visualization

The next word cloud shows the most frequent words present in the description column from the train data set. Most frequent words are displayed in a bigger font.

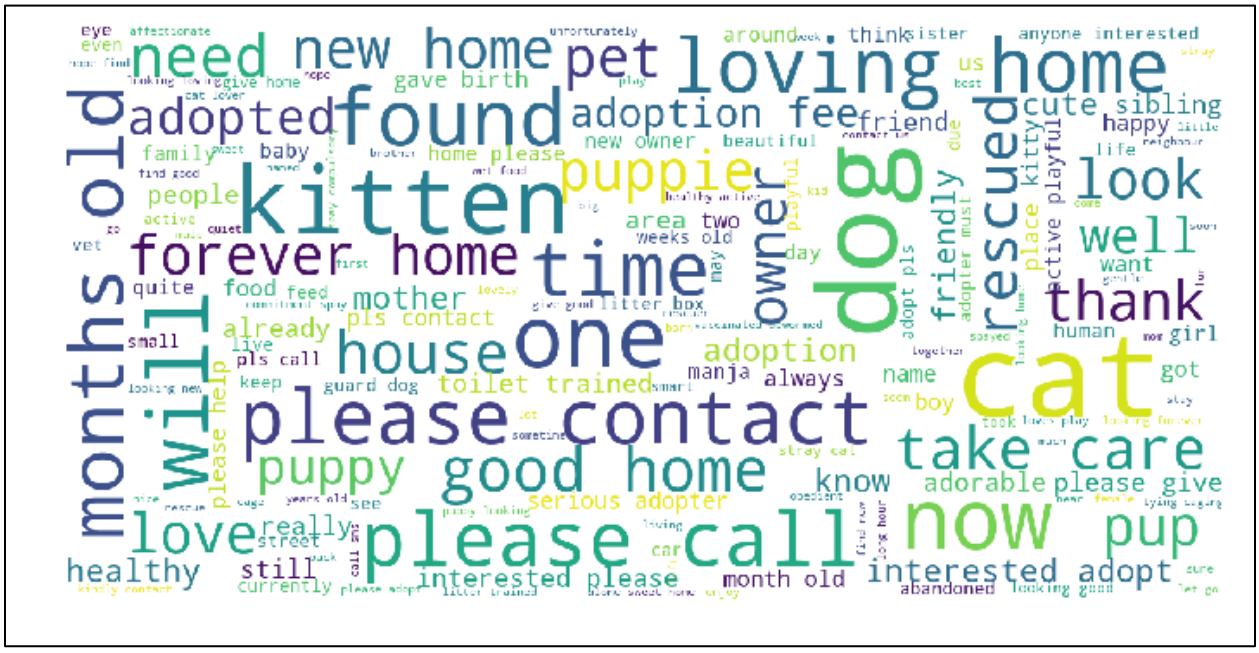


Figure 9 Word Cloud - Pet Description

It is interesting to note the relation between words and the age of the pet. Common words are kitten, puppy, pup, puppie, kitty which would come in the profile if the pet is young. However, there would never be words related to old pets.

As I have described, age is one of the most important features related to the adoption, I would advise to definitely include these words in the description of young pets, and words like healthy, adorable or friendly in the description of older pets.

Reflection

The process that I followed in this project was:

1. I searched and interesting problem which I could solve by applying the techniques learned in this course.
2. Found PetFinder competition. I liked the purpose and the framework related to this problem.
3. Analyzed the data by following the next steps:
    a. Variable identification – Categorical, continuous
    b. Univariate Analysis – Create density plots, boxplots and violin plots
    c. Bi-Variate Analysis – Create a pair plot
    d. Outlier identification – Crate boxplots
    e. Variable transformation – Applied TFIDF over text
4. Create benchmark model.
5. Experiment with different classification algorithms
6. Tune the hyper-parameters of the algorithms
7. Evaluate and define a final model.

Steps 6 and 7 were the most difficult and time consuming. Especially because they required a lot of research to understand the algorithms and the good practices to improve their performance. Hyper-parameter tuning takes a lot of time because it requires to experiment with different values until the optimal set is found. Applying techniques like grid search often take one to several hours for it to finish. For example, I applied grid search to random forest and even with 4 concurrent workers it took 1 hour.

The most interesting aspect of the project was to validate some hypothesis found during data analysis by observing which were the most important features of the final model. I liked that through the process I learned several new techniques found in kernels. I am happy to work on problems that have a positive impact.

Improvement

I identified 3 interesting improvements:

1. I found a technique which was being used by several kaggle competitors. It consisted on a way to use regression scores an generate the classification labels by finding optimum

thresholds[20] . To find these thresholds the author proposed and optimizer which took the predictions, the target variables and minimized the Kappa loss function. An improvement to my current XGBoost model could be to change the task to regression and apply this optimization technique.

2. I could have applied grid search for XGBoost parameter tuning. By applying this process, I could have found more optimal values for certain parameters. Time was the main restriction to apply this technique.
3. Create two models, one to classify cats and the other for dogs. Each model could have learned specific attributes of each pet type and then apply both models to generate the result.

There is potential experiment with other techniques an improve my final model, the winners of the competition reached a score of 0.46613.

[20] How to use regression here?