Background

Python machine learning scripts have been generated to predict the life expectancy of cats in years given certain parameters of the cat. A model is trained by inputting a csv file into the python training script. Then after the model has been completed predictions can be made from that model using the python prediction script.

Working Environment

Tools

- Macbook - macOS 10.13.3
- Pycharm IDE - professional (includes code coverage)
- Jupyter Notebook
- Python 3.6.x

Packages

- Pandas
- Numpy
- Sklearn
- Xgboost

Working Instructions

Training Model

To train the prediction model simply use your terminal and navigate into the PredictiveHireChallenge folder and run the script train.py:

python3 train.py

At this point please provide the file path to your input csv file containing the cats data. The columns of the cats.csv file that the script is expecting is as follows:

- age at death (numeric and in years)
- breed (string)
- date of last vet visit (string timestamp)
- hair length (numeric)
- height (numeric)
- number of vet visits (numeric)
- weight (numeric)

Where age of death is the target column.

Prediction

To predict the cats life expectancy in years you have to also provide a csv file but this time leaving out the target column which is “age at death”. The command to run the script is:

python3 predict.py

And the csv file that is expected includes these columns:

- breed (string)
- date of last vet visit (string timestamp)

- hair length (numeric)
- height (numeric)
- number of vet visits (numeric)
- weight (numeric)

After prediction has been run with the generated model the predictions will be merged with your inputted csv file in the **output** folder.

Overall Methodology & Explanation

Below is a list of steps (in order) taken to prepare the dataset for model training, will explain in more depth for each of these points later on.

1. Removing duplicates from the data
2. Splitting data into a training and validation set (80/20 split)
3. Simplifying date of last vet visit column to include only the year (as time stamp is to the second)
4. Convert all data to float data type (from strings etc)
5. Replacing NULL or NaN values in the dataset with median of the respective column
6. Replacing outliers of each column (will explain in more detail below)
7. Normalise the data set (each column contain only values from 0-1)
8. Create training model using XGBoost
9. Perform 10 fold cross validation to tune model hyper-parameters
10. Create final training model using best parameters from tuning and export Model.

Preprocessing

As all data scientists know most of the work involved in a machine learning project is data preprocessing to fit the “correct data” into the machine learning algorithm. We will explain each preprocessing step and why it was necessary to do it.

Removing Duplicates

Removing complete copies of each data entry or duplicates was necessary to not mislead the model. First the duplicates are likely to be erroneous so must be removed and second removing duplicates will make sure model trained are not more favourable towards a certain outcome.

Convert all data to float data type

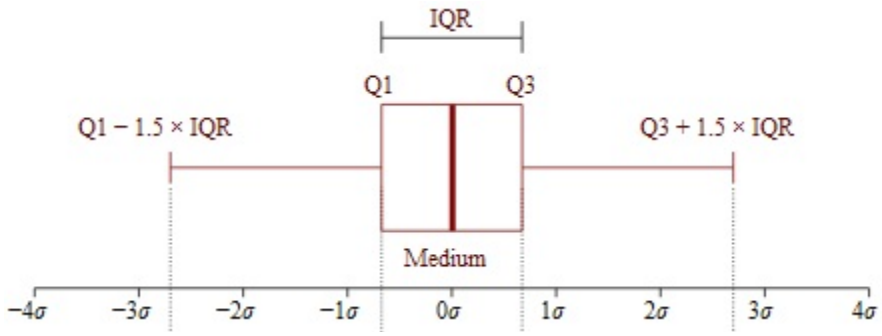
This step was simply necessary because machine learning algorithms will only take in numerical data fields and not strings. So conversion from strings or other data types to float was necessary.

Replacing NULL or NaN values

Replacing null values was necessary because otherwise we cannot normalise the dataset. Also because replacing NULL or NaN values will possibly improve model performance. Replacing with median value of the specific column and not average because if there are outliers the average will still be very misleading. Median gives a more accurate representation of the data column. There is also another reason, once strings have been converted to float their value is coded for each string character, replacing with median will simply give the most occuring string whereas the average will possibly invent a non existent string.

Replacing outliers from each numeric column

Now there are no such thing as outliers for a string column. Also removing and replacing outliers is very important for model performance also it is better for normalisation. If outliers will not removed it is possible the normalisation will make very small values very close to 0 and the outlier close to 1 and the model may lose accuracy.



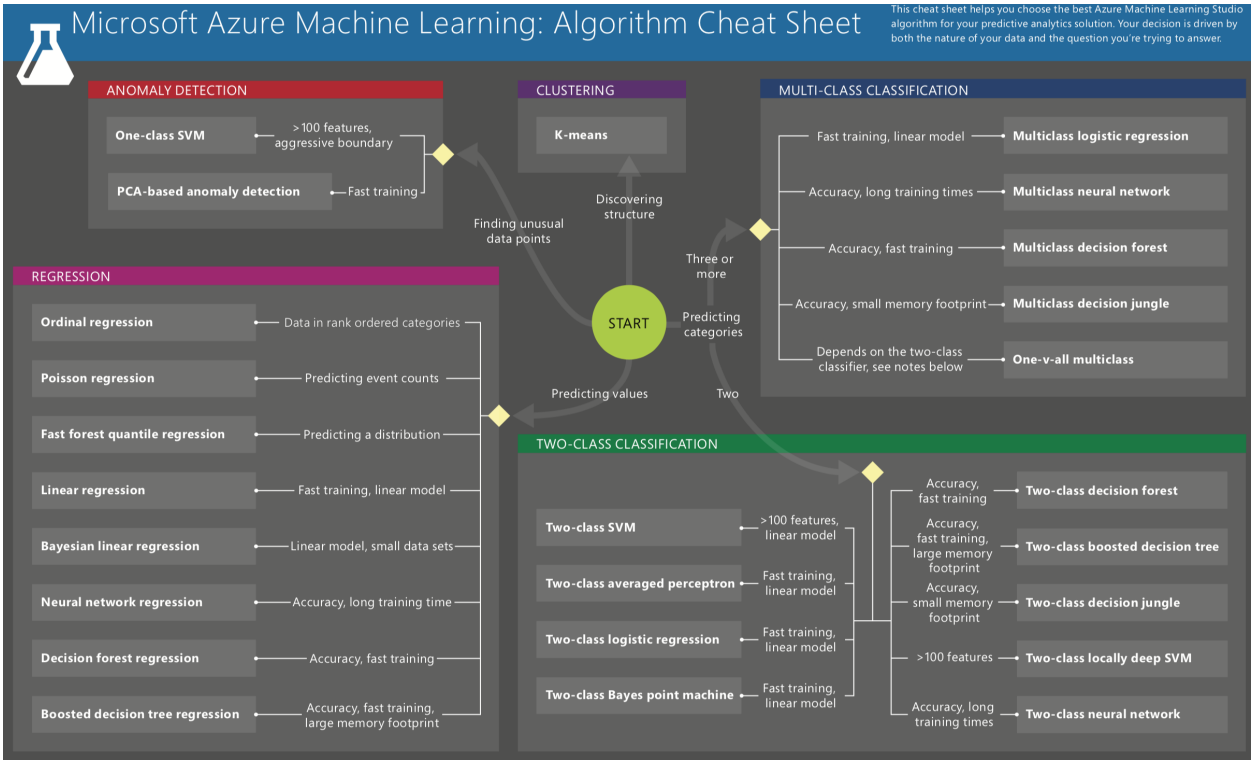
The outliers were defined as anything that is above or below 1.5*(IQR - interquartile range)

This is a very established stats concept. And the maximum outlier was replaced with the 95th percentile while the minimum outlier was replaced with the 5th percentile.

Normalise data set

Strictly speaking normalising dataset might not be necessary however, usually within this step if there were skew present this is where we can correct the data skew. Nevertheless, there have been studies and accuracy tests which suggest normalising the dataset is more friendly for the machine learning model as when they create hyper dimensions the values of each column are more closely linked (from 0-1) so this may improve accuracy of the model.

Model Selection & Training Concept



Choosing a 80/20 using the Pareto Principle which states that 20% of the dataset usually covers 80% of the dataset's variation so a validation set of 20% of the dataset was sufficient.

The XGBoost machine learning model was used because the problem was a regression problem as opposed to multi-class classification. From the handy cheat sheet above from microsoft, for high accuracy and fast training in regression we could choose either decision forest or boosted decision tree. Now as XGBoost (extreme gradient boosting) is a form of boosted decision tree it required large memory footprint and because there is no limitation in that front, XGBoost was selected due to its higher accuracy than other solutions.

Code Quality

Pycharm uses best practice code standards and will allow user to quickly correct code to adhere to standards. Pycharm checks indentation, unused variables/functions classes etc, static methods, class self initialisation etc etc.

Another big part of code quality can be proven through code coverage tests and reports generated. Unit tests were written for each python script, through code coverage it can be easily seen how many "lines of code" were covered by the unit tests and will give you a percentage of coverage.

The idea is to first cover all functions written in the scripts with unit tests then if more quality is desired then agree on a percentage of code coverage (a common one is 80% of all code should be covered).

You can see the code coverage reports in the **coverage** folder and open up the index.html file. In there you can see each of the python scripts and the amount of code covered in each.
For this coding challenge I have not covered all functions as it is only a sample of what it can do and how to prove code quality. I have started writing unit tests for pre-processing functions. They are stored in the **tests** folder.