Bark-Paw-Pagation

Anish Katariya
Southampton University
ak7n14@soton.ac.uk

Mikolaj F Bak
Southampton University
mfb2g15@soton.ac.uk

ABSTRACT

The project aims to explore different machine learning techniques in order to predict certain outcomes from a given dataset. We go through the machine learning process of data exploration, data cleaning, feature engineering and extraction, and comparison of different models. The dataset chosen was taken from a Kaggle competition that aims to predict the how fast a pet will get adopted.

ACM Reference Format:
Anish Katariya and Mikolaj F Bak. 2018. Bark-Paw-Pagation. In *Proceedings of Southampton Advanced Machine Learning (Southampton'2018).* ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

The Kaggle competition hosted by PetFinder.my proposed the idea of being able to predict how fast pets can get adopted based on their advertisements. The company provided a set of training and testing data containing metadata of the pet as well as the photos that were uploaded associated with each pet. The adoption speed values range from 0-4 with 0 being the fastest and 4 meaning the pet was not adopted after 100 days of listing. For most of the advertisements there is a written description of the pet that was shown in the advertisement. Each of these descriptions was fed into a Google Natural Language API to produce a sentiment data file associated with it, highlighting the sentiment and magnitude scores as well as important sentences. All of the images provided were also processed by Google Vision API that aimed to give their prediction of what the image contains as well as other metadata such as dominant colours and position of objects on the image.

2 DATA EXPLORATION

The training set provided included 14993 advertisements. All ads were labeled with the appropriate adoption speed, 96% of them provide sentiment data files from description and 98% of them have metadata from their images. In order to try to understand the relationship between this data we decided to see if there were any clear correlations between the current features and the adoption speed. As shown in Figure 1, none of the initial features provided any strong magnitude of correlation. The most significant features from there were Age, Breed1, FurLength, and Vaccinated.

The next experiment we did in order to determine feature importance was feeding this initial set of features into a Gradient Boosting model in order to get a plot of feature importance scores. Figure 2 shows this plot and highlights the importance of the Age and Breed1 features. More surprisingly the amount of photos submitted for an advert is in the higher end of importance whereas the type of pet (dog=1, cat=2), did not seem as important.

Looking at the distribution of Age along the data, we found some clear patterns and outliers present. Since the age feature is captured in months, a lot of the values were multiples of 12 indicating that

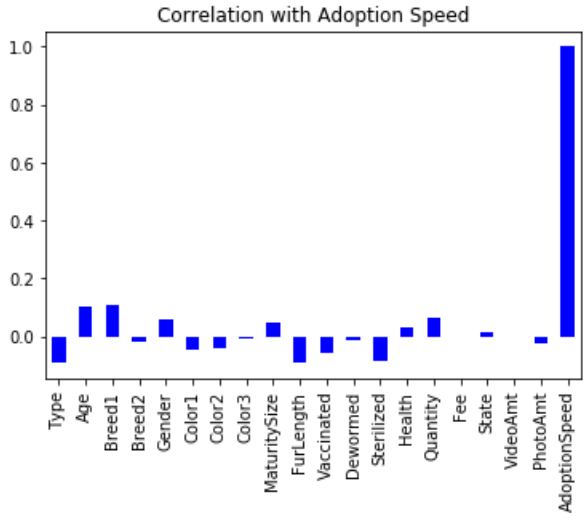

Figure 1: Initial Correlation with Adoption Speed

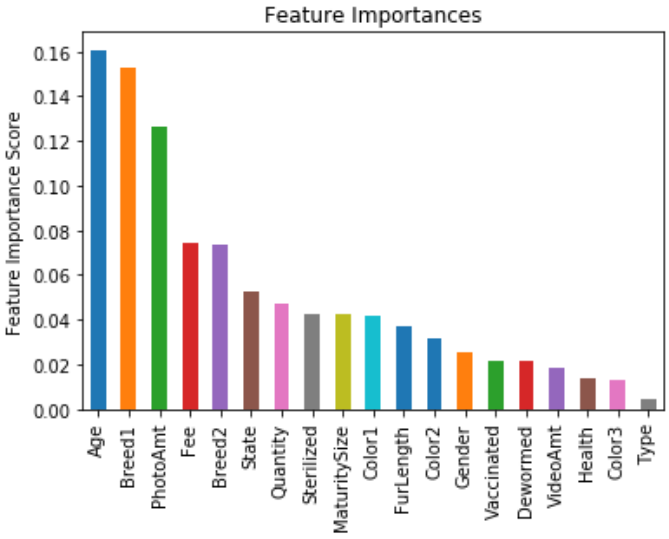

Figure 2: Initial Feature Importance

the owner was simply estimating the age and was not exact. The plot also indicated that there were cases where a pet was listed with an age of 200+. Although cats are known to live longer, we investigated further and realised that the inputs from the users in the age category did not match what they inputted as description. One dog was listed to be 212 months old, but the description indicated that

the dog was 2 1/2 years old. This shows the need of detection of outliers during data cleaning to help training our model.

2.1 Baseline Model and QWK

To get the initial idea of how the data would behave with our models, we decided to test a weighted random classifier and check it's accuracy. This classifier would randomly pick a class for a given data-point with preference for more common classes. As expected this yielded a bad result with the accuracy of 24% (±1%), however we can use this to determine whether other models are able to learn or are picking random values. Another metric that we have employed in assessing the quality of our models was Quadratic Weighted Kappa [3] (QWK), which compares two "raters" and determines whether they are both in total agreement (QWK = 1), total disagreement (QWK = -1), or the raters are unrelated (QWK = 0). This was also the value used to compare the models in the original Kaggle competition. Since this model was classifying the data at random, it received a QWK score of 0.00 (±0.02).

3 DATA CLEANING

3.1 Numerical Features

When dealing with NaNs with numerical features the options are to replace the value of NaN with another value or to completely get rid of that row of data. Because in this case there is no clear correlation of features to adoption and because we suspect the description and images provided with the pets have high value, we decided not to get rid of any rows because of NaN values. There were cases where the NaN value was replaced with the mean such as sentiment score and magnitude since there were not many values missing. The rest of the instances we filled the values with a -1 if they were NaN. Although filling features with -1 values can have a negative effect on the importance, we used Scaling, Logging, and Binning as techniques to help turn our data into a more useful format for our model.

3.1.1 Logs and Scaling. As discussed before, the Age feature introduced the notion that the training data has outliers that need to be dealt with. A way to minimise the impact that the outliers will have is to take the log of all the values of that feature. This way features with very high values wont be negatively impacted.

In addition to taking the log of the feature, scaling also helps to clean the data. All the initial features in the training set have different ranges that they cover and if not scaled the model could tend to pay more attention to those higher values. Scaling will help gradients converge quickly and for the model to learn weights more efficiently.

3.1.2 Binning. Binning was used in only one of the numerical features. The Fee data provided ranged from 0 to 3000. The reason behind binning this particular feature after taking the log and scaling it was that we wanted to capture the effect that the ranges of fees have on the adoption speed. Since the fee can be a floating point value and most of the values are 0, then it makes more sense to take the price ranges as a feature instead of the scaled value.

3.2 Categorical Features

The categorical features were already encoded to a number that corresponded to a value in one of the CSV files provided by Kaggle. We decided to use One Hot Encoding in order to represent such features. The reason for using this over Label Encoding is that for features with many values label encoding will assign large values which might affect the importance the model gives to these features. An example would be the Health feature ranges from 1-4, where 1 is best health and 3 is seriously injured, but value 4 means not specified. If left as a Label Encoded feature, the model would assume 4 corresponds to the worst health when it should not, hence the need to one hot encode it. For the State and Breed features, since they had a large amount of categories, it would be space and time inefficient to one hot encode the values. Instead of one hot encoding these features, we used a separate numerical feature to represent them. For State, we used the AvgIncome and PopDensity features since they were constant for a given state, and for the Breed categories we used an engineered feature named Breed Popularity which will be described in Section 4.3.

As well as the given categorical features, the addition of new features from data cleaning also needed to be encoded appropriately. During the numerical cleaning, we felt necessary to add new columns when certain features had NaN across the row. An example of this is the has_metadata feature indicating whether the advert had any metadata related to the pictures. The value for this columns was True or False and therefore the need to label encode it.

3.3 Text Features

The two text features we focused on were the Name and Description. For the Name, there were some normal values but most of the times the Name values were the breed of the dog, a NaN value, or another word that indicated they had no name. In order to deal with this variations, we filled the NaNs with 'None' and searched through the field for any value that was similar to 'None' to signify no name and added a has_name feature that shows this.

The Description feature often was a paragraph describing the pet or set of pets. The NaN values on this feature were simply replaced by the value '<Missing>'. This will help when turning the Description text into a set of features determining the importance of the words used.

4 FEATURE ENGINEERING

4.1 Outside Sources of Data

In an attempt to capture features that could better represent how fast a pet would get adopted and due to the low correlation of the data given even after data cleaning, we decided to look at outside factors. The first idea we had was to look at the effect of each state on the Adoption Speed. The first new feature we collected were the population sizes, state size, and average income. The initial intuition was that states that have a higher population density and are wealthier are more likely to have families that would want to adopt a pet. Unfortunately, the new PopDensity and AvgIncome features extracted from [1] did not improve our model or showed indication that they had any correlation with Adoption Speed. This

is in part due to the fact that 90% of the ads belong to the three biggest states so its hard to learn difference between adoption speed of the smaller states.

The second outside source of data dealt with two of the most important features, Age and Breed. Since we had seen that the Age inputs can contain some outliers or incorrect data, we decided to extract the average life expectancy for every breed of dogs and cats in order to estimate the months to live that the pet has. If the dog was a mixed breed, the average life expectancy of the two breeds was taken into account when computing months to live. If a pet was simply 'Mixed Breed' as their given breed, the average expectancy of all pets of that type was used. This new feature also helped us to do further data cleaning since we could see that if the pet was said to have negative months to live then the inputted age was incorrect. We allowed for 2 years buffer and so we got rid of 10 ads that were clear outliers. Although not the most important feature, months to live proved to be better than all of the features previously given.

4.2 Description

For the given Description feature we needed to be able to extract features using some natural language processing technique. We decided to use TFIDF in order to capture the importance of the words chosen in each add compared to all the descriptions given. Since these technique would give us too big of a vector (14993, 221209) to add to the model, we decided to reduce its dimensionality using SVD with a fixed number of components. The number of components is meant to maximise the variance of the data some time constraints. We decided to implement a function that could give us an indication of the optimal amount of components to achieve at least 0.95($\pm$0.05) variance. This proved time consuming and as seen in Figure 3 the number of components had little effect on variance, but the trend we saw was that as components increased the variance increased. We tried 10, 15, and 20 as number of components all of which ended up being in the top 30 important features when using Gradient Boosting, but had equal impacts on accuracy score.

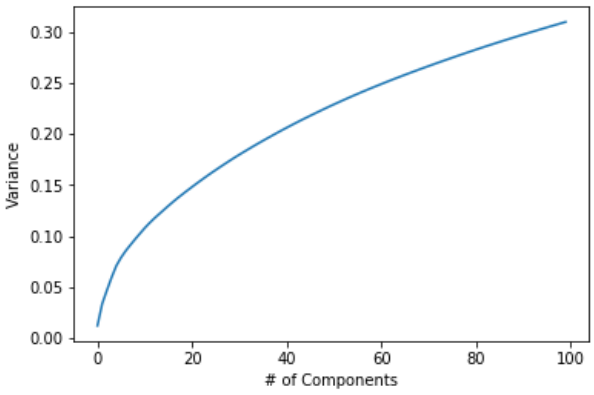

Figure 3: Effect of Number of Components on Variance

4.3 Breed Popularity

Since Breed was shown to be an important feature in our first model, we explored a way to better capture its effect on Adoption Speed. Given the training data we decided to rank the popularity of each breed based on how fast the breeds were getting adopted. We grouped the ads by Breed1 and Breed2 and gave different scores based on the adoption values 0-4. For fast adoption speeds we awarded positive popularity votes and for slower speeds we gave negative adoption votes. The votes were averaged by number of times the breed was used in the ads. The score ranged from -2 to 2 with 2 being the most popular. This new feature was one of our best features and was able to increase our accuracy score when using Gradient Boosting by around 2-3%. Its correlation with Adoption Speed was the highest out of all features.

4.4 RescuerID Count

We were able to count the number of times each Rescuer was included in the dataset and used this as a feature. Although trivial to generate, this feature is worth mentioning because it became the most important feature used with Gradient Boosting. The use of this feature increased the accuracy of the model by 3-4%. What is worth mentioning is that even though we were excited at the results produced by this feature, we were not confident that this was a good feature to learn when determining adoption speeds. The number of rescued pets a person might have should not determine how fast a pet should be adopted since a new rescuer could have a very desirable pet, but because it is the first rescue our model could disregard other important features of the pet.

5 FEE

5.1 Fee Prediction

One feature we were surprised about was the Fee. With our domain knowledge of dog adoption, we predicted that it would be quite high on the list of priorities for the potential dog adopter. However, when we see Figure 1, fee has a correlation of around 0% with adoption speed.

However, we believed that Fee feature was created differently to the other features. The Fee was assigned to each pet, after all the other features had been collected by the adoption company. We first ran a Gradient Booster with default parameters, with all the features previously mentioned as the data, and the Fee as the labels.

The problem with this was that it showed the model was unable to predict higher fee values. The model fee prediction ranged from 0 - 16, compared to the 0 - 3000 in the real data. This happened because the majority of the real Fees were 0, as so the model would be penalised during training when it predicted a high Fee incorrectly. The other problem was that the model did not predict 0 for any of the Fees, but a number close to it.

To solve these problems we decided to use the binned Fees as labels, and use a Gradient Booster Classifier.

The model this created allowed us to see feature importance of each feature for determining the Fee, as shown in Figure 4. Interestingly, a few of the features which had little impact on the adoption speed have a large impact on the Fee, namely Health, Vaccinated and Dewormed. This explains their low impact on Adoption Speed, as the better a dog is in terms of health, the higher the fee would be set.

Unfortunately, this model did not predict the Fee well. It predicted that all the Fee were in category -1 (Fee = 0). The reason this

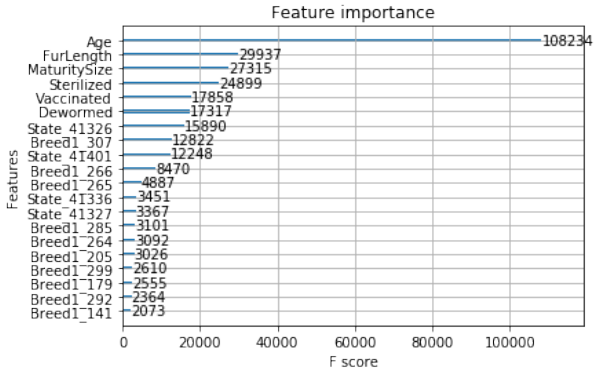

Figure 4: Most important features for the predictions of Fee

happened was because the data was extremely unbalanced, as you can see from Figure 5, 84.5% of the training data was in category -1. The problem with the model was that the default parameters are max_depth = 3 and n_estimaters = 100. This means that the tree is not big enough and/or the training time not long enough, for it to be beneficial for the model to try and categorise one of the smaller categories.

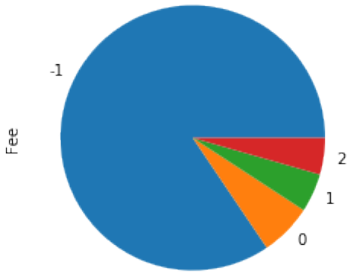

Figure 5: Binned Fee, number of records per category

To fix this we increased both of these parameters, however this has the negative that it could cause over-fitting, and takes longer to run.

With this new model we were able to predict better Fee values. The results are shown in Figure 6, and how well it performed in a number of metrics shown in Figure 7. These were done on 20% of the data not used in the training phase. The high results on the testing data, leads us to believe that this model has not over-fitted. To make sure this was not a outlier, we tested this 5 times and found the average difference in precision per class, this is shown in table 1. The high variance on these metrics, mean that the model is changing more that we would want depending on the seed value given.

We then subtracted the actual Fee from our predicted Fee. This gave us a feature, that would tell us if the pet was available to be adopted for less or more than would normally be asked.

Predicted / True	-1	0	1	2	All
-1	2498	11	11	17	2537
0	116	44	1	0	161
1	86	2	67	2	157
2	50	0	5	89	144
All	2750	57	84	108	2999

Figure 6: Confusion Matrix for the prediction of binned Fee

	precision	recall	f1-score	support
-1	0.91	0.98	0.94	2537
0	0.77	0.27	0.40	161
1	0.80	0.43	0.56	157
2	0.82	0.62	0.71	144

Figure 7: Performance on various metrics of Binned Fee Prediction

Class	Average Precision	Precision Variance
-1	0.88	(±0.01)
0	0.62	(±0.07)
1	0.63	(±0.16)
2	0.65	(±0.13)

Table 1: Table showing how the Precision changes depending on the seed variable

When this feature was used with our main Gradient Booster to predict adoption speed, it improved the QWK score to 0.52 (±0.03) from 0.48.

5.2 Adjusted Fee Prediction

This predicted Fee, was created by finding out how the adoption company calculates a suitable fee. However it does not take into account if the company has been undervaluing or undervaluing certain features.

We can see that the company is probably overvaluing the Health Feature. The health feature is a number 1 to 4(1 = Healthy, 2 = Minor Injury, 3 = Serious Injury, 4 = Not Specified), with the higher the number the worse the injury, and as we can see from Figure 1, the correlation for adoption speed is positive, therefore the greater the injury the faster the adoption speed. One explanation is that the company is charging a higher fee than they should for a healthier pet.

To take account for this we created another feature, which took the binned Fee, and we subtracted one if the adoption speed was

zero and added one if the adoption speed was four. We then created the adjusted predicted fee, the same way we made the predicted fee, but with the updated binned Fee feature.

When added to our main Gradient Booster, it improved the QWK score to 0.525 (±0.002) from 0.52. This had a small impact, but this was expected as this feature and predicted fee were very co-linear, and therefore most of the information in this feature was in the last feature.

6 IMAGE FEATURES

For the image features the first step was to make sure that all the image features were of the same size. To do so the center of the images were recognised and the images were shaped into squares of similar size.

Initially our approach was to take sift features from images and form a bag of visual words using k-means clustering. However due to the massive amount of noise present in each image, also due to the fact that in some cases, images had features of multiple animals made this approach unfeasible for our use.

Our second approach was to then match sift features in cases

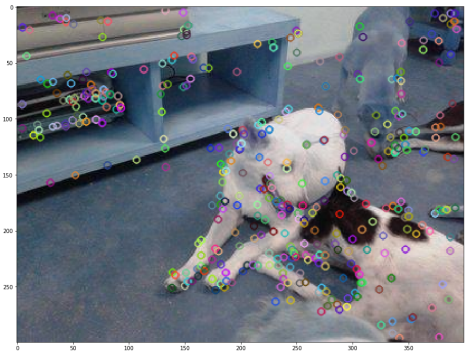

Figure 8: Sift features of images capturing lot of noise

where we were given multiple images of the same animal and only use features that it was able to match across images. However this approach too failed as in most cases there was only one picture available for the animal and in cases where multiple images were available, there was hardly any change in the angle of the images taken so sift feature matching was still able to match noise between the images. Finally we decided upon using a convolutional neural

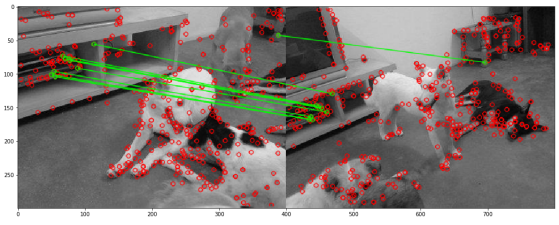

Figure 9: sift features still matching noise

network in order to extract image features. A pre-trained dense121 network trained on dogs and cat images was found and transfer learning was used in order to get selected features from this network. A 1024 dimensional vector of each image was produced by this network.

In order to get more training data and also to achieve angle invariance, we also flipped every image present in the training data and put it through the dense121 network in order to get more features.

Training on images took our machines took a long time, as a result one of the machines having a NVidia gpu was used to access the cuda libraries present in pytorch to get GPU acceleration and speed up the process.

7 MODEL COMPARISON AND TUNING

7.1 Multilayer Perceptron Model

7.1.1 Architecture. The network architecture that we decided on was a feed-forward network with two hidden layers with a ReLU activation function and an output layer with a softmax activation. Categorical cross-entropy loss function and Adam optimisation algorithm [4] were used for training of the network. For the number of neurons in each hidden layer, the number of input nodes was taken into consideration and the number was set to a value between $\frac{2}{3}$rds of the input nodes and the number of input nodes. This bound helps with achieving generalisation and avoiding over-fitting the model.

7.1.2 Results without dropout. Firstly, the network was tested without using dropout [5]. This meant an increased risk of over-fitting, but could give us an early indication of the model's capabilities. Figure 10 shows that the network was able to increase it's accuracy over the training set, however it did not manage to carry over that performance to testing on the validation set. By performing stratified K-fold cross validation, we found that the model's accuracy on the training set was equal to 43% (±3%). Unsurprisingly, the accuracy was better than random guessing, which is also evident in the QWK score which was equal to 0.39 (±0.04). Training this model took on average 15 minutes to complete while using GPU accelerated version of TensorFlow backend.

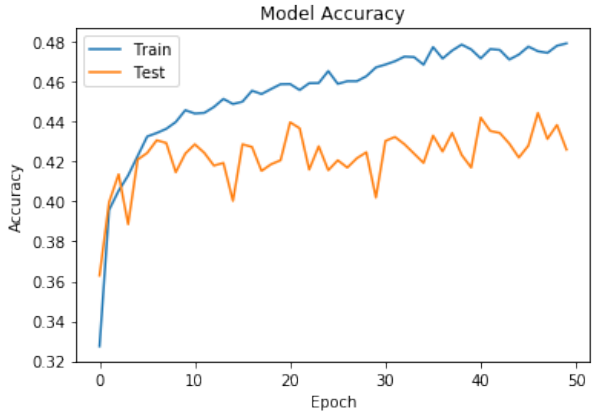

Figure 10: One of the runs of the neural network shows lack of generalisation and poor performance in the validation set.

7.1.3 Results with dropout. To try and combat the over-fitting of the model to the training data, dropout was employed as a prevention method. Dropout reduces complexity of the network, leading to less over-fit. Figure 11 shows that, while accuracy doesn't appear to improve, the performance on the train set resembles the performance on the validation set. The accuracy of the model has not improved as it remained at 43% (±1%), however the lower error shows that model is less over-fitted compared to the network without dropout. The QWK score decreased slightly to 0.38 (±0.04). As with the model without dropout, training took around 15 minutes to complete.

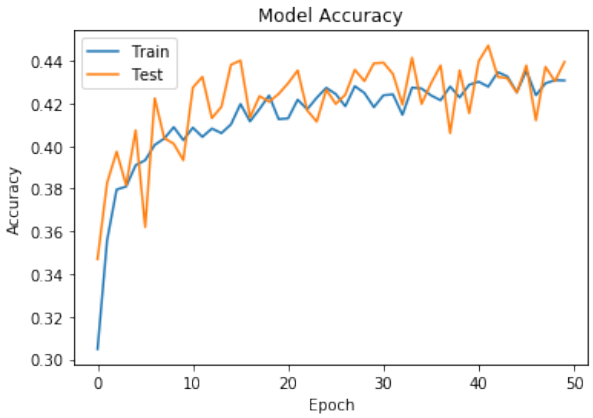

Figure 11: One of the runs of the neural network shows results on the train and validation sets are much closer.

7.2 XGBoost Model

Another model that we decided to test was gradient boosting. This was achieved by using the XGBoost [2] library for Python. Using the same data that was used for training the neural network we managed to achieve much better results. The model worked by firstly treating the problem as a regression problem and then using the regression model with learned bins to classify the data point to a class. This approached gave as the QWK score of 0.52 (±0.03) on the training set, which is comparable to the performance of top classifiers from the Kaggle competition. A large problem with our model was the data being unbalanced. This is most clearly seen in the class 0, that represents pets that were adopted within a week. In the training test there are only 410 such cases, where other classes have around 3000-4000 samples each. This lead to the model being incapable of recognising that class. Training this model took around 5 hours, much longer compared to the neural network models.

8 CONCLUSION

Through the use of common machine learning techniques as well as comparison of different models, we aimed to achieve a prediction on the speed of adoption for a pet dataset. The best results we achieved were through XGBoosting when using QWK as the score metric. Our model proved to have a score similar to the top-10 scores in the Kaggle competition for the training dataset. The data cleaning and feature engineering stages proved more effort for this dataset since there was not much correlation between the initial data and the adoption speed. We used data from outside sources to find correlations as well as using current data to predict Fee as a measure of comparison. We also used pre-trained machine learning models like transfer learning through a dense121 network to extract image features. Comparing neural network to XGB showed that our neural network tended to overfit which is the main reason why XGB had a better score. The project allowed for comparison of different machine learning techniques and gave us a better understanding on the process of trying to give meaning to the data given for classification.

REFERENCES

[1] 2019. Department of Statistics Malaysia. https://www.dosm.gov.my/
[2] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2* (2015), 1–4.
[3] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin* 70, 4 (1968), 213.
[4] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.