COMP9417 Group Project:
Pet Adoption Speed Prediction

Min Jung Son (z5204803) | Victor Tsang (z5209633)
Kurt Wang (z5207982) | Jinhao Huang (z5207964)

Kaggle link - https://www.kaggle.com/c/petfinder-adoption-prediction

1 Introduction

Across the world, millions of stray animals and unwanted pets are left without homes and forced to suffer. With the rise of technology, animal adoption over the internet has become increasingly popular, and in turn, the opportunity to use big data and machine learning to help animal shelters has also grown.

Petfinder.my is a leading animal welfare platform in Malaysia, with a database of over 150,000 animals. They have identified that animal adoption rates are strongly correlated to the metadata associated with their online profiles, such as listing attributes, descriptions, and photo/videos. In this Kaggle competition, Petfinder has provided tabular data, JSON metadata, description sentiment data, as well as video and image data from their database of adoption listings. The challenge is to use this data to develop algorithm(s) to predict the adoptability of pets; specifically, predicting the adoption speed of pets.

In this project, we aim to use the tabular dataset and experiment with a variety of models to classify pets' adoption speed into one of five classes:

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. (There are no pets in this dataset that waited between 90 and 100 days).

As such, our problem is a **multi-class classification problem**, and the evaluation metric is the **quadratic weighted kappa**, as specified by the Kaggle competition. This evaluation metric is described in more detail later in this report.

2 Implementation

2-1 Model Selection

From our research, we have selected 7 models that were suitable to our task of multi class classification: Logistic Regression, Support Vector Machines (SVM), Gaussian Naive Bayes, Multinomial Naive Bayes, Decision Trees, Random Forest, and Gradient Boosted Model (GBM and XGBoost). Since open-source implementations of these models are readily available, we directly test all of them in our model algorithm selection process and compare their performances. In our evaluation, we prioritise achieving a higher Quadratic Weighted Score, the scoring metric designated by the Kaggle competition organisers. That being said, we also consider the training time and complexity of each model. Our hypotheses about our shortlisted models are thus oriented around the trade-off between performance and cost.

One of the candidate models in consideration is Logistic Regression, selected for its computational efficiency, its applicability to multiclass predictions and the availability of hyperparameters such as the inverse of regularization strength (C) and class weights. We hypothesised that logistic regression would perform better than some models such as Naive Bayes because of these characteristics; however, we also hypothesised that logistic regression may not perform as well as more complex learning methods such as tree-based methods since it relies on average or no multicollinearity between independent variables and can only learn a linear decision surface.

We also considered SVM as an appropriate model. It aims to create a margin-maximising hyperplane to separate the pet data into the five classes. SVM are effective in handling multiple features of pet data and are relatively robust to outliers as the hyperplane is only affected by support vectors (data points closest to the hyperplane). In this study, we consider the simplest Linear kernel and the Radial Basis Function (RBF) kernel. Kernels are vital in transforming the original pet dataset to define an optimal decision boundary. The Linear kernel linearly separates data points. We hypothesise an improved performance of the RBF kernel for non linear datasets as it projects the high dimensional data and then searches a linear separation for it.

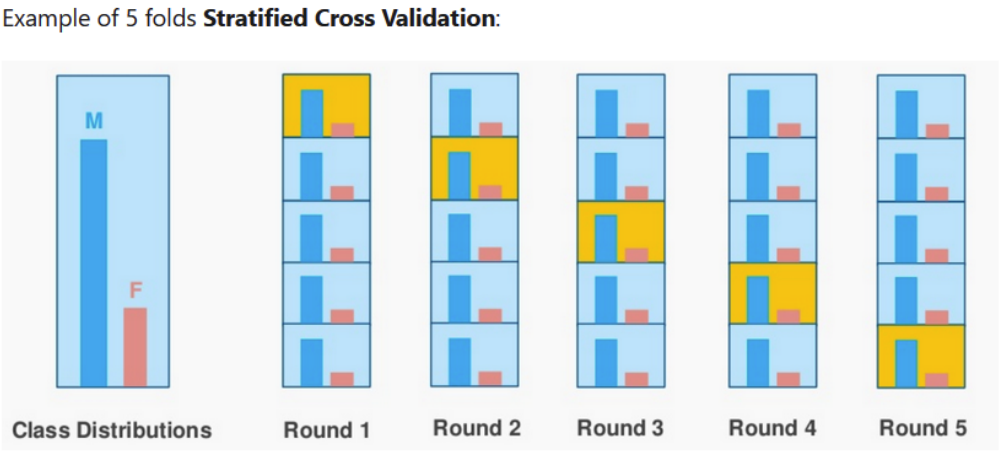
Next, we select two different Naive Bayes classifiers, specifically the Gaussian and Multinomial Naive Bayes classifiers, based on Bayesian Statistics using prior and posterior probabilities. The Gaussian model follows a normal distribution and supports continuous data while the Multinomial model uses term frequencies of discrete features to perform classification. Although Naive Bayes classifiers can handle both numeric and categorical data making them a good baseline model with little feature engineering and hyperparameter tuning required, one key issue is the assumption of independence of features, which is rarely true for real-world data. Furthermore, these models commonly come across the “Zero Frequency Problem”, an issue where certain levels of a categorical variable are present in the testing dataset but not present in the training dataset, thus causing a zero probability to be assigned. Our hypothesis for these

Naive Bayes models is that they will perform decently and quickly, however there are more suitable and flexible models such as gradient boosted trees.

Lastly, existing literature on classification problems also suggest Decision Tree implementations as suitable models that inherently handle both categorical and continuous features. There are further Decision Tree implementations with ensemble methods that we can use to improve and validate performance. In particular, we have chosen Random Forest and gradient boosting methods (regular gradient boosting, and XGBoost). These ensemble methods generally have good out-of-the-box performance but do have their own shortcomings that can be mitigated with careful feature selection and hyperparameter tuning. Our hypothesis is that gradient boosting methods will have the best performance due to their flexible nature and evaluation idiosyncrasies - however this will come with a greater computational cost.

2-2 Cross validation

Cross Validation is the gold-standard procedure for estimating the generalized performance of a model. In this project, we used Stratified K-Fold Cross Validation as our evaluation method. The underlying principle behind cross validation is to divide our dataset into K folds, using k-1 folds to train our model and evaluating performance on the remaining folds. By repeating this process with K different evaluation folds, we are able to approximate our out-of-sample performance. However, the random nature of cross validation means there is a possibility that the folds will be unrepresentative of the whole dataset's response's distribution, introducing bias into the model - for example, if a class is only present in a single fold, generalization may not be able to be achieved. Stratified K-Fold Cross Validation is thus a method of solving this problem, shuffling the dataset before splitting it into K-folds such that each class is equally represented across all test folds. A diagram is describing Stratified K-Fold Cross Validation is shown below:



2-3 Hyperparameter Tuning

A hyperparameter is a parameter that is set before the learning process begins - as such, they are tunable and often directly affect how well a model performs. These hyperparameters differ from model to model, and have been summarised in appendix (6-4) and will be explored in depth later on.

In our implementation of our models, we relied on Scikit-learn's implementation of a grid search with cross-validation (*GridSearchCV*). This performs an exhaustive search over the given hyper-parameter space, training and evaluating the machine learning model with a different combination of hyperparameters each time.

2-4 Evaluation Metric

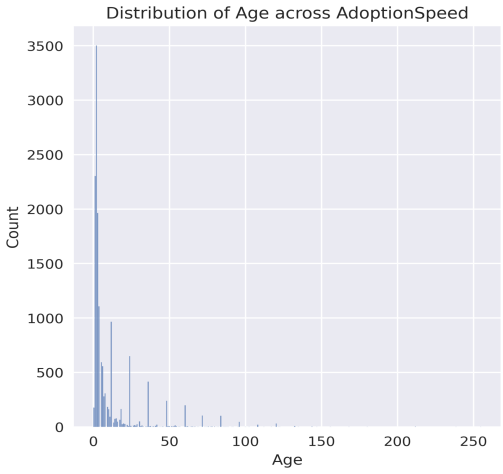
Models are evaluated based on the Quadratic Weighted Kappa (QWK) score, which takes a value between -1 and 1. This ratio is calculated between predicted and actual scores. A negative quadratic weighted kappa implies that the model is worse than random, whilst a perfect prediction will yield a score of 1.

Our results have 5 possible categorical classifications: 0,1,2,3,4. To calculate the quadratic weighted kappa, first a confusion matrix O is constructed such that $O_{i,j}$ corresponds to the number of adoption records that have a rating of *i* (actual) and *j (*predicted). A matrix of weights W, is calculated based on the difference between actual and predicted classes. Finally, a matrix of expected classification E, is calculated by taking the outer product between the actual rating's histogram vector of classes and the predicted classes' histogram vector of classes. This is normalized such that E and O have the same sum.

In our study, we implement this Quadratic Weighted Kappa through the sklearn.metric.cohen_kappa_score function in Python with weights='quadratic'.

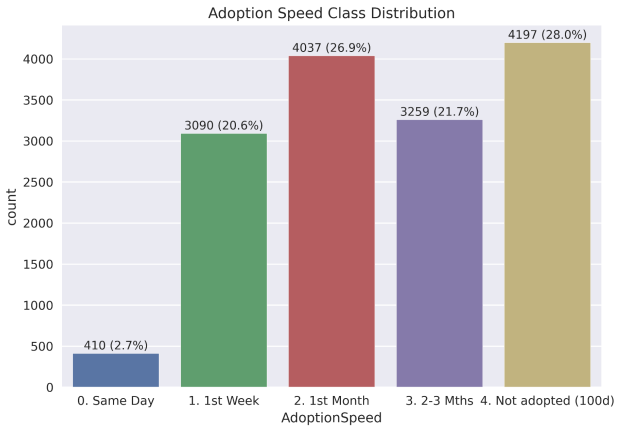
3 Experimentation

3-1 Exploratory Data Analysis



The Petfinder.my dataset consists of tabular, image, image metadata, and sentiment data. To avoid spending excessive time on data cleaning and wrangling, we decided to strictly use the tabular dataset for our experimentation. The tabular data includes a labelled training dataset and an unlabelled test set, containing 14,993 and 3,972 observations, respectively. The tabular dataset has 24 variables, including descriptive features the gender of the pet ('Gender'), whether the pet is a dog or a cat ('Type'), the specific breed(s) ('Breed1' and 'Breed2'), and health information such as vaccination status ('Vaccinated'), whether it has been sterilized ('Sterilized'), and other such information. A data dictionary of all the columns has been included in the appendix (6-1).

A significant proportion of the variables are also categorical. Of the 25 variables, 17 are categorical while 5 variables are numeric and 3 are text ('PetID', 'RescuerID', and 'Description'). Notably, our numeric variables are relatively right-skewed: for example, over 50% of the training set have age of 0-3 months, with the other 50% ranging from 3-255 months. This can be seen in the plot on the right.

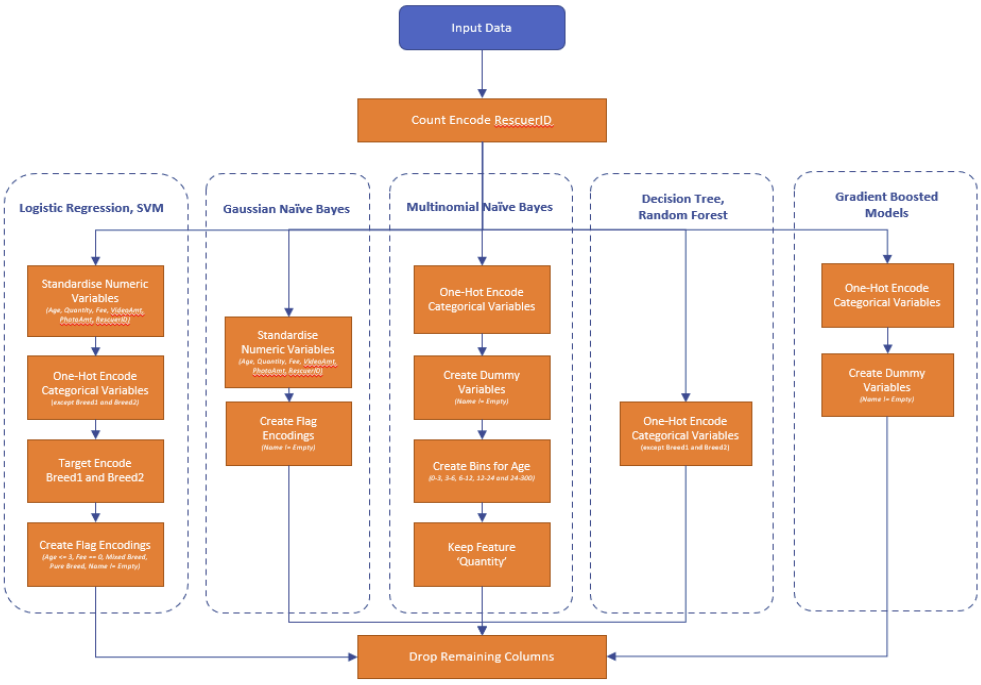


A summary of the class distribution is shown on the left. We see that the labels are relatively balanced, except for label 0 (pets adopted on the same day they are listed) and only represents 2.7% of the training data. As a result, we hypothesise that models that can account for this imbalance may perform better, such as Gradient Boosting Models.

3-2 Feature Engineering

Preprocessing and feature engineering is necessary for our models because the input data should be engineered such that it represents the real-world relationships, thus improving performance. Furthermore, each learning method has its own unique traits, meaning they may work more optimally with different representations of the same data. A table summarising the techniques and justifications for each is included in the appendix (6-5).

A pictorial representation of different final feature engineering pipelines is seen below, with more detailed explanations of our experimentation following:



In all our feature engineering processes, PetID, RescuerID and Description were dropped as they are text labels and therefore not applicable to our models. However, in the case of RescuerID, the data was count-encoded, with the reasoning that previous adopters would be more likely to adopt another pet going into the future. Beyond this step of processing, our feature engineering pipelines began to diverge.

For logistic regression and SVM, existing literature has supported the use of one-hot encoding to encode categorical features that did not have high cardinality - this included: Type, Gender, Health, FurLength, State, Color1, Color2, Color3, Sterilized, Vaccinated, Dewormed, and MaturitySize. However, for high-cardinality categorical variables, we avoided exploding the number of features (which negatively affects convergence as a result of the curse of dimensionality) and used Target Encoding. Breed1 and Breed2 had over 300 different levels - for each of these labels, we calculated the mean of AdoptionSpeed in the training set, and used these as encodings. However, for tree based implementations all categorical data were one-hot encoded as the model would be rendered invalid if splitting occured on non-ordinal categorical data in an ordinal manner, since colours are not ordinal variables. Next, since we are performing regularization in Logistic Regression and SVM, it was important that we also standardized our numeric variables such as Age, and Quantity. We then created flag encodings to flag important information - for example, pets younger than 3 months are often much more attractive than older pets, and free pets are much more appealing to adopters. Similarly, pure breeds and mixed breeds are very popular in Malaysia, so flags were created for these features. Finally, we dropped the remaining columns, resulting in 68 input features.

For our Naive Bayes models, different feature engineering methods were required for Gaussian and Multinomial models. The Gaussian Naive Bayes classifier is broadly applicable to a wide range of data and robust to noise, and thus does not require much feature engineering to begin with. We still improved the data by standardizing the same numeric variables as mentioned above as well as removing outliers. A dummy variable for Name was also added for the same reasons as with logistic regression and SVM. On the other hand, for Multinomial Naive Bayes classification we require one-hot encoding of all categorical features: Type, Gender, MaturitySize, Vaccinated, Dewormed, Sterilized, Health, FurLength, State, Breed', Breed2, Color1, Color2 and Color3. This was necessary as the model calculates term frequency for the features, which is much easier when each categorical feature is separated into its own column. The same dummy variable for Name was applied, and the Age column was separated into bins of 0-3, 3-6, 6-12, 12-24 and 24-300. Apart from the features mentioned, only the column Quantity was retained in the data as it fit the term frequency requirements of the model.

3-3 Models Selection/Tuning

After performing feature engineering for each model, we began our model selection process. This involved running a grid search over our hyperparameter search spaces with cross validation. We then compared each model's training score and test score (which was obtained by making predictions on the unlabelled input data and submitted to Kaggle for the Private Leaderboard score). We summarise our model selection search in a table on the next page, and our results in the next section.

As mentioned in the **Implementation** section, we initially selected Logistic Regression for its efficiency and extensibility using hyperparameters. First, we tested a simple baseline model with no adjustments and no changes to the default hyperparameter settings. After establishing this baseline, we observed the model was not converging during training, suggesting that feature engineering was necessary. We then performed the aforementioned preprocessing and feature engineering, followed by a grid search with stratified 5-fold cross validation over the hyperparameters 'C' and 'class_weight' using the L1 norm for the penalty. Due to the high computational cost of an exhaustive search, we used a parameter grid consisting of C values increasing logarithmically from 0.0001 to 100. A logarithmic search was used to compromise for the expensive nature of a grid search, as the marginal effect of changes in the regularization strength are not very significant. We also tested the class_weight parameter, experimenting with either no balancing of the response classes or placing greater emphasis on less frequent classes to compensate for fewer observations. We believed balancing the class_weights would be useful, as our EDA showed that class 0 is much less frequently observed than other classes. Moreover, we opted to use L1 penalty (in other words, performing a LASSO regression) as we observed significant multicollinearity in the dataset - for example, Health, Dewormed, and Fee are all related to each other, which makes intuitive sense given that healthier animals are also likely to be dewormed, and these additional factors can contribute to a higher fee.

Table of Hyperparameters

Model	Hyperparameters	Search Space	Effect
Logistic Regression	C	0.0001, 0.001, 0.01, 1, 10, 100	Controls the inverse of regularization strength, and affects the degree of regularization. A smaller value specifies stronger regularization
	class_weight	None, Balanced	Controls the weights associated with the response's classes. "None" assigns an equal weight to all classes, whereas balanced mode places greater emphasis on less frequent response classes.
Support Vector Machine (SVM)	C	0.1, 1, 10, 100, 500, 1000	Controls the inverse of regularization strength, and affects the degree of regularization. It trades off misclassification of training examples against the simplicity of the hyperplane.A smaller C value creates a smooth decision surface, whilst a high C aims at classifying all training examples.
	Gamma	1, 0.1, 0.01, 0.001, 0.0001	Gamma defines the influence of a single training example. The small value gamma will loosely fit the training dataset whilst a high value of gamma will exactly fit the dataset, causing over-fitting.
	Kernel	Linear, RBF	Kernels transform the original pet dataset to define an optimal decision boundary. The Linear kernel linearly separates data points. The RBF kernel projects the high dimensional data and then searches a linear separation for it.
Gaussian Naïve Bayes	var_smoothing	100 Values from -9 to 0 evenly spaced w.r.t. the log scale.	Portion of the largest variance of all features that is added to variances for calculation stability.
Multinomial Naïve Bayes	α (alpha)	0.01, 0.1, 0.5, 1.0, 10.0	Uses Laplace smoothing to solve the zero frequency issue of Naïve Bayes.
Decision Tree	Max Depth	1-10	Controls the depth of the tree. This is needed to control model complexity as to prevent overfitting.
	Max Features	1-63	Controls the number of features to consider when splitting. Also prevents overfitting (or underfitting).
Random Forest	Max Depth	5-50	Controls the depth of each tree. This is needed to control model complexity as to prevent overfitting.
	Max Features	‘auto’, ‘sqrt’, ‘log2’	Controls the number of features to consider when splitting. Also prevents overfitting (or underfitting).

	N estimators	100-800 (increment:100)	This controls the number of trees that are fit before taking the final ‘vote’. More trees mean your model will improve (or at least not degrade) but computation time may take much longer.
Gradient Boosted Classifier	N estimators	1-200	This controls the number of sequential trees fit – prevents overfitting. Needs to be tuned with respect to learning rate.
	Max Depth	1-10	Maximum depth controls overfitting, tuned in combination with CV.
	Learning Rate	0.01-1.0	The learning rate controls the contribution of each tree to the final model. Lower values are generally preferred to increase robustness and prevent overfitting.
XGBoost	Max Depth	2-8	Maximum depth controls overfitting, tuned in combination with CV.
	Learning Rate	0.01-0.5	The learning rate controls the contribution of each tree to the final model. Lower values are generally preferred to increase robustness and prevent overfitting.
	Subsample	0.5-1.0	Denotes the proportion of observations to be randomly sampled for each tree. Lower values make the algorithm more conservative but values that are too small may lead to underfitting.
	Min Child Weight	1-5	The minimum sum of weights of all observations required in a child. Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
	Col-sample by Tree	0.5-1.0	Similar to Max Features this hyperparameter denotes the fraction of features to be randomly sampled for each tree.
	N estimators	50-300 (increment:50)	Controls number of boosting rounds (or trees) and hence degree of fitting by the model.

Similarly, we selected and began experimentation on the SVM with a baseline model of a linear kernel and a default hyperparameters value of C=1.0. We then ran the SVM model with a RBF kernel, where optimal hyperparameters of C and Gamma were chosen through the same GridSearch cross-validation outlined above. Whilst the time taken to train and fit RBF-kernel SVMs were longer, there was a substantial improvement compared to the baseline, even without any encoding of categorical variables and additional feature engineering. We applied one hot and target encoding to process nominal categorical variables. We continued the use of grid search cross-validation for hyperparameter tuning of the SVM. As this search has a high computational demand, we compromised by restricting our parameter grid to C values [0.1, 1, 10, 100, 500, 1000] and Gamma values [1, 0.1, 0.01, 0.001, 0.0001].

Like the other models mentioned so far, the two different Naive Bayes models were selected due to their suitability to the database in different ways. Firstly, the Gaussian Naive Bayes model's ability to handle both numerical and categorical data made it extremely easy to run the raw data through without drastic feature engineering required. Thus, the baseline model was already robust. On the other hand, the Multinomial Naive Bayes model did not seem like an obvious selection based on the raw data and had a weak baseline model. However, given it's fair use and effectiveness in other models such as with Logistic Regression and SVM, the data was feature engineered to be a great fit for using the Multimodal model as well since it was able to use term frequency. For the two Naive Bayes models, we applied the same grid search and cross validation as our previous models,using the evaluation metric to measure performance. The results can be seen below, with a fairly small var_smoothing value for Gaussian and a low level of smoothing required with alpha=0.1 for Multinomial. This suggests that the train/test splits for our data were good representatives of the entire dataset, as well as the fact that the data did not have extreme variations post feature engineering.

Furthermore, we experimented with four different tree based implementations: Decision Trees (simple), Random Forest, Gradient Boosted Classifier (simple), XGBoost. The Simple Decision Tree classification method was used as a baseline model for predictive accuracy as it was the most simple tree implementation. This was improved upon by ensemble methods of Random Forest as well as gradient boosting methods, which were chosen due to the complex nature of our data set where single Decision trees may be extremely prone to overfitting, as small anomalies in the data could completely alter how a tree is built and hence affect predictive accuracy. Random Forests mitigate both these issues by minimizing error due to both bias and variance – with its power lying in its ability to limit overfitting without significantly increasing error due to bias. A further step to this is to use gradient boosting classification methods which aim to improve on some shortcomings of Random Forest models albeit having pros and cons of its own. The two main reasons we selected to continue further experimentation with gradient boosting methods was that these models offered better predictions for highly unbalanced data and Random Forest models were inherently biased towards categorical variables with a large number of classes. Both of these reasons were clearly identified as issues prevalent within our database, and thus allowed gradient boosted models to be more optimised than Random Forests.

To further optimise our model, we performed the same hyperparameter tuning with grid search and cross validation as with the previous models. We first determined a theoretically plausible range for each hyperparameter. Then we tested a few combinations on each model to determine whether they truly impacted model performance before running the full grid search, cross validation method mentioned above. A summary of hyperparameters can be found in the tables in the following pages:

3-4 Results

After implementing and performing GridSearch with cross validation, we arrived at the following results for each respective model:

Table of Results

Model	Best Hyper Parameters	Baseline Score	Best model training score	Best model test score (Private Leaderboard)
Logistic Regression	C: 10 penalty: L1 class_weight: None	0.1620	0.3215	0.2590
Support Vector Machine	C:10000 Gamma:0.0001	-0.0142	0.4373	0.2309
Gaussian Naive Bayes	Var_smoothing: 3.5111917342151277e-07	0.1706	0.2016	-
Multinomial Naive Bayes	alpha:0.1	0.0365	0.2570	0.2520
Decision Tree	criterion': 'gini', 'max_depth': 5, 'max_features': 40	0.1817	0.2572	-
RandomForest	bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2', 'n_estimators': 300	0.2292	0.2581	-
GBM	learning_rate': 0.1, 'max_depth': 3 'n_estimators': 200	0.2574	0.2920	-
XGBoost	colsample_bytree': 0.5, 'learning_rate': 0.06, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.7	0.2597	0.3114	0.3233

The results for the Logistic Regression model indicate that our model may be overfitting, as our model test score is significantly lower than our training score, even after cross validation.That being said, these results support our hypothesis that logistic regression may have relatively good performance as a result of the hyperparameters that allow tuning for regularization and class weights. However, a possible reason for our performance may be that our training set may not be a representative sample of the real distribution -- despite class 0 being significantly less frequent than the other classes, gridsearch determined no class balancing to be necessary.

In examining the results of SVM models, our results support the hypothesis that RBF-kernels outperformed quadratic kappa scores of baseline linear-kernel SVMs. In fact, quadratic kappa scores of the baseline was -0.0142 whilst the RBF with optimal parameters (without encoding or feature engineering) resulted in 0.2121. With one-hot and target encoding of categorical features on the training dataset, this led to the highest quadratic kappa score of 0.4373. The SVM with a RBF-kernel, optimal hyperparameters and encoding was determined as the SVM submission, producing a competition result of 0.2309. The decreased performance on the test samples indicates an overfitting of the SVM to the training dataset.

From the summary table above, we can see that the Multinomial Naive Bayes model with one-hot encoded categorical features performed better with a Quadratic Kappa Score of 0.2570, while the Gaussian model was 0.2026. This suggests that using one-hot encoding in feature engineering was a success, allowing the data to be fitted accurately on the Multinomial model and thus generating better results. Other reasons for this conclusion could be the lack of modifications, detail and tuning specific to our dataset in relation to the Gaussian model in comparison to the Multinomial model. However, these models are not optimal for the dataset and have worse performance than the XGBoost model. One key reason for this may be that the conditional independence assumption in all Naive Bayes models is clearly wrong in this situation, where we have many features such as Vaccinated and Dewormed that are strongly correlated or may be dependent. Another example of this break of conditional independence is MaturitySize and Age.

For tree implementations, the simple decision trees performed the worst but surprisingly there was quite an increase in Quadratic Kappa Score between Random Forest and the gradient boosting models with ~4% increase in score as opposed to our initial hypothesis that they would perform somewhat similarly. As our ultimate aim is to find the best overall model for submission to the Kaggle competition, the XGBoost model which performed the best in training with a score of 0.3114 was ultimately submitted with a final Quadratic Kappa Weighted score of 0.323 (3.d.p) received from the competition. (Full tree implementation results in appendix (6-2)).

4 Conclusion

The aim of this assignment was to predict adoption speeds for pets given physical attributes and traits listed on their online listing profiles. After researching and experimenting with a variety of models, both parametric and non-parametric, we have identified some key takeaways about our results and potential improvements going forward.

The most significant challenge we faced was the problem of overfitting in the model selection and tuning process. From our results, we see that every model except XGBoost and Multinomial Naive Bayes had a much lower score on the test set compared to the training set. Some contributing factors include the complex nature of the data, most likely being of a non-linear nature, making less complex models that rely on strict assumptions such as logistic regression or SVM less reliable. Moreover, non-parametric models such as tree models and gradient-boosted models are able to learn a non-linear decision surface, meaning they are better able to capture the true nature of the relationships behind the data. We also identified that feature engineering made a significant difference in improving performance, suggesting that more fine-tuned or fit-for-purpose processing techniques may yield even better results going forward.

Moreover, the imbalanced response classes that we identified during our exploration caused problems in the learning process, as our training set may not be representative of the full population, and thus bias can appear in models. Again, non-parametric methods, careful feature engineering, and more appropriate learning approaches can mitigate the issue of class imbalance. Our final major learning point is that although Grid Search is an exhaustive technique, due to its high computational cost we were only able to search a narrow range of values. As such, a randomized or more “intelligent” search method may yield better hyperparameters for dealing with overfitting and improving our overall scores.

5 References

Biau, G. (2012) Analysis of a Random Forest Model. Journal of Machine Learning Research, 13, https://www.jmlr.org/papers/volume13/biau12a/biau12a.pdf

Bradley, J., Rajendran, S. (2021) Increasing adoption rates at animal shelters: a two-phase approach to predict length of stay and optimal shelter allocation. BMC Vet Res , 17(70), https://doi.org/10.1186/s12917-020-02728-2

Chen, T., Guestrin, C. (2016) XGBoost: A Scalable Tree Boosting System. ArXIV Labs, 1603(02754), https://arxiv.org/abs/1603.02754

Fawagreh, K., Gaber, M. M., Elyan, E. (2014) Random forests: from early developments to recent advancements. Systems Science & Control Engineering, 2(1), https://doi.org/10.1080/21642583.2014.956265

Hu, T., Song, T. (2019) Research on XGboost academic forecasting and analysis modelling. Journal of Physics: Conference Series, 1324(12091), https://iopscience.iop.org/article/10.1088/1742-6596/1324/1/012091/meta

King, G., Zeng, L. (2001) Logistic Regression in Rare Events Data. Political Analysis, 9, https://j.mp/2oSEnmf

Lewis, M. (2015) Graphing grid scores from GridSearchCV. Stack Overflow. https://stackoverflow.com/questions/37161563/how-to-graph-grid-scores-from-gridsearchcv

Mendel, A. (2019) Naive Bayes: A Baseline Model for Machine Learning Classification Performance. KDNuggets. https://www.kdnuggets.com/2019/04/naive-bayes-baseline-model-machine-learning-classification-performance.html/2

Miller, J. (2020) Creating a Good Analytics Report. Kaggle. https://www.kaggle.com/jpmiller/creating-a-good-analytics-report

Nazrul, S. S. (2018) Multinomial Naive Bayes Classifier for Text Analysis (Python). Towards data science. https://towardsdatascience.com/multinomial-naive-bayes-classifier-for-text-analysis-python-8dd6825ece67

Navlani, A. (2018) Naive Bayes Classification using Scikit-learn. DataCamp. https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn

Neely, M. N., Van Guilder, M. G., Yamada, W. M., Schumitzky, A., Jelliffe, R. W. (2012). Accurate Detection of Outliers and Subpopulations With Pmetrics, a Nonparametric and Parametric Pharmacometric Modeling and Simulation Package for R. Therapeutic Drug Monitoring, 34(4), https://doi.org/10.1097/FTD.0b013e31825c4ba6

Powers, D. (2015) Understanding stratified cross-validation. StackExchange. https://stats.stackexchange.com/users/21922/david-m-w-powers

Ren, Q., Cheng, H., Hai Han, H. (2017) Research on machine learning framework based on random forest algorithm", AIP Conference Proceedings 1820(80020), https://doi.org/10.1063/1.4977376

Rosinki, W. (2019) Baseline Modelling. Kaggle. https://www.kaggle.com/wrosinski/baselinemodeling

Saerens, M., Latinne, P., Decaestecker, C (2002) Adjusting the Outputs of a Classifier to New Priori Probabilities: A Simple Procedure. Neural Comput, 14 (1), https://doi.org/10.1162/089976602753284446

Scikit-learn (2011) Gaussian Naive Bayes. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

Scikit-learn (2011) Linear Models. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/linear_model.html

Scikit-learn (2011) Logistic Regression. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Scikit-learn (2011) Multinomial Naive Bayes. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

Scikit-learn (2011) Metrics. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

Scikit-learn (2011) Receiving Operator Characteristic. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

Scikit-learn (2011) Support Vector Classification: SVC. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/svm.html#classification

Scikit-learn (2011) Support Vector Machines. Scikit-learn: Machine Learning for Python. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Shin, T. (2020) A Mathematical Explanation of Naive Bayes in 5 Minutes: A thorough explanation of Naive Bayes with an example. Towards data science. https://towardsdatascience.com/a-mathematical-explanation-of-naive-bayes-in-5-minutes-44adebcdb5f8

Tatman, R. (2019) Six steps to more professional data science code. Kaggle. https://www.kaggle.com/rtatman/six-steps-to-more-professional-data-science-code

Yadav, S., Shukla, S. (2016) Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification. IEEE 6th International Conference on Advanced Computing, 10(1109) https://ieeexplore.ieee.org/abstract/document/7544814

6 Appendix

6-1 Data Dictionary

1. **PetID** - Unique hash ID of pet profile
2. **AdoptionSpeed** - Categorical speed of adoption. Lower is faster. This is the value to predict. See below section for more info.
3. **Type** - Type of animal *(1 = Dog, 2 = Cat)*
4. **Name** - Name of pet *(Empty if not named)*
5. **Age** - Age of pet when listed, in months
6. **Breed1** - Primary breed of pet *(Refer to BreedLabels dictionary)*
7. **Breed2** - Secondary breed of pet, if pet is of mixed breed *(Refer to BreedLabels dictionary)*
8. **Gender** - Gender of pet *(1 = Male, 2 = Female, 3 = Mixed, if profile represents group of pets)*
9. **Color1** - Color 1 of pet *(Refer to ColorLabels dictionary)*
10. **Color2** - Color 2 of pet *(Refer to ColorLabels dictionary)*
11. **Color3** - Color 3 of pet *(Refer to ColorLabels dictionary)*
12. **MaturitySize** - Size at maturity *(1 = Small, 2 = Medium, 3 = Large, 4 = Extra Large, 0 = Not Specified)*
13. **FurLength** - Fur length *(1 = Short, 2 = Medium, 3 = Long, 0 = Not Specified)*
14. **Vaccinated** - Pet has been vaccinated *(1 = Yes, 2 = No, 3 = Not Sure)*
15. **Dewormed** - Pet has been dewormed *(1 = Yes, 2 = No, 3 = Not Sure)*
16. **Sterilized** - Pet has been spayed / neutered *(1 = Yes, 2 = No, 3 = Not Sure)*
17. **Health** - Health Condition *(1 = Healthy, 2 = Minor Injury, 3 = Serious Injury, 0 = Not Specified)*
18. **Quantity** - Number of pets represented in profile
19. **Fee** - Adoption fee *(0 = Free)*
20. **State** - State location in Malaysia *(Refer to StateLabels dictionary)*
21. **RescuerID** - Unique hash ID of rescuer
22. **VideoAmt** - Total uploaded videos for this pet
23. **PhotoAmt** - Total uploaded photos for this pet
24. **Description** - Profile write-up for this pet. The primary language used is English, with some in Malay or Chinese.

Source: Petfinder Adoption Prediction Kaggle Competition (https://www.kaggle.com/c/petfinder-adoption-prediction/data)

6-2 Tree Model Performances

Model	Feature Selection/ Engineering	Hyperparameters	Quadratic Kappa Score (3.d.p)	
			CV Score	Test set score (conducted on 30% subset of original training data)
Simple decision tree	*None*	*{'criterion': 'entropy', 'max_depth': 6, 'max_features': 17}*	*0.311*	*0.319*
Simple decision tree	Baseline	'criterion': 'gini', 'max_depth': 5, 'max_features': 40	0.257	0.187
Simple decision tree	Advanced	'criterion': 'gini', 'max_depth': 5, 'max_features': 40	0.229	0.196
Random Forest	*None*	*'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2', 'n_estimators': 300*	*-*	*0.340*
Random Forest	*Baseline (top 4 important, without one-hot encoding)*	*'bootstrap': True, 'criterion': 'gini', 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 800*	*-*	*0.296*
Random Forest	Baseline (with one-hot encoding)	'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2', 'n_estimators': 300	0.258	0.274
Random Forest	Advanced	'bootstrap': True, 'criterion': 'gini', 'max_depth': 9, 'max_features': 'auto', 'n_estimators': 100	0.253	0.249
Gradient Boosting Classifier	Baseline	'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200	0.292	0.228
Gradient Boosting Classifier	Advanced	'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 32	0.279	0.247
XGBoost	Baseline	'colsample_bytree': 0.5, 'learning_rate': 0.06, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.7	0.311	0.241

**Feature selection = none means no feature selection/engineering whatsoever e.g. no one-hot encoding.*

***Feature selection = baseline means only initial alterations have been enacted e.g. one-hot encoding.*

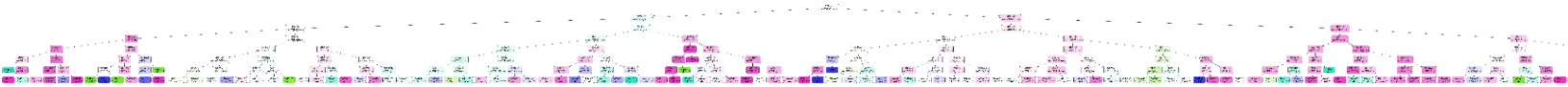
****Feature selection = advanced means getting rid of unimportant variables/variables with high cardinality.*

*****Models in italics are invalid due to non-one hot encoding issues on non-ordinal data. Please refer to the Feature Engineering, Decision Trees section for more details.*

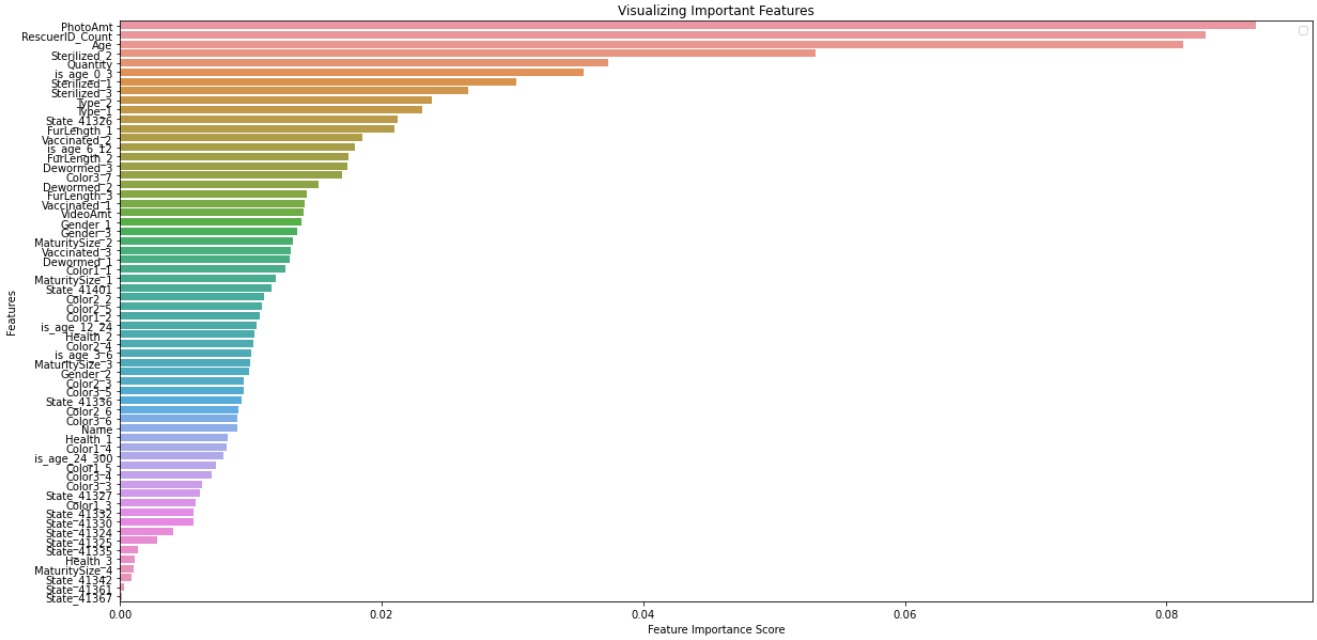
6-3 Visualisation of Tree

Initial decision tree - low performance, lots of overfitting and barely interpretable results due to the size of the tree.

Decision tree, with some tuning. Still faces many of the problems listed above.



Visualisation of important features used for pruning in Random Forest models:



6-4 Hyperparameter Definitions

Model	Hyper Parameters
Logistic Regression	● C: Inverse of Regularization Strength ● penalty: vector norm (l1 or l2) ● class_weight: emphasis placed on the classes
Support Vector Machine	● C: Inverse of regularisation strength. ● Gamma: defines the influence of a training example.
Gaussian Naive Bayes	● var_smoothing: Portion of largest variance of all features that is added to variances for calculation stability.
Multinomial Naive Bayes	● alpha: Controls Laplace Smoothing which solves the Zero Frequency Problem.
Decision Trees	● Max_depth: maximum depth the tree is allowed to grow to ● Max_features: maximum number of features considered when splitting on a node
RandomForest	● Max_depth: maximum depth the tree is allowed to grow to ● Max_features: maximum number of features considered when splitting on a node ● N_estimators: number of trees fit before taking final ‘vote’
Gradient Boosted Model	● N_estimators: number of sequential trees fit ● Max_depth: maximum number of features considered when splitting on a node ● Learning rate: controls the contribution of each tree to the final model
XGBoost	● Max_depth: maximum depth the tree is allowed to grow to ● Learning rate: controls the contribution of each tree to the final model ● N_estimators: number of boosting rounds ● Subsample: denotes proportion of observations that are randomly sampled for each tree ● Colsample_bytree: denotes number of features to be randomly sampled for each tree ● Min_child_weight: The minimum sum of weights of all observations required in a child

6-5 Feature Engineering Technique Summary

Preprocessing/ Feature Engineering	Description	Reasoning
One-Hot Encoding	For each unique label, create a binary dummy variable	Since labels are numeric, one-hot-encoding ensures that the model does not falsely learn a relationship between the non-ordinal labels and the target.
Target Encoding/ Mean Encoding	For each unique label, calculate the mean of the target variable. The encoded values will be the target means for each group.	In scenarios where there is a large number of levels to a categorical variable, one-hot encoding causes the number of dimensions to explode. Target encoding is a method of encoding the relationship between each label and the target.
Ordinal Encoding	Re-order the labels according to the mean of the response. The encoded labels will be from 0 to (n-1) where n is the number of labels.	The original dataset had some veterinary and health features ordered arbitrarily. By re-labelling these features, we can convert it into an ordinal encoding relative to the mean of the response.
Count Encoding	For each unique label, count the number of occurrences in the training set. The encoded values will be the count for each label.	In scenarios where there is a large number of levels to a categorical variable, one-hot encoding causes the number of dimensions to explode. Count encoding aims to use the number of observations belonging to each group as a proxy for the categories themselves.
Standardisation	Subtract the mean and scale to unit variance.	Regularisation places constraints on the coefficients - to make a valid comparison between coefficients during optimization, the coefficients of numeric variables need to be standardised.
Dummy Variables	Create a binary variable indicating a certain condition.	Certain attributes would be significantly more important than others, e.g. Age <= 3 months or Fee == 0.