07/10/25

Project 01 Report

Abhiram Reddy

To prepare the data, I first loaded the given CSV from the class repository and standardized a few headers so they matched what the project requirements. For example, I changed Animal ID to AnimalID and Outcome Type to OutcomeType. I then dropped duplicate rows, then parsed the date fields (DateTime, Date of Birth, MonthYear), and finally created simple time features from DateTime: OutcomeYear, OutcomeMonth, OutcomeDOW, and OutcomeHour. I decided to convert Age Upon Outcome strings like "2 years" or "3 weeks" into a single numeric feature called AgeAtOutcomeDays, then I split Sex Upon Outcome into two clearer variables, Sex_simple (male, female, unknown) and Intact_status (intact, altered, unknown). I also added a HasName flag, which captures whether the animal had a name. Looking at the instructions, I restricted the dataset to the two classes of interest Adoption and Transfer. I imputed the missing values, removed non-predictive columns, and very importantly dropped Breed before one-hot encoding so I did not overpopulate the feature space with thousands of Breed columns. Finally, I one-hot encoded the remaining categorical columns to build a clean, fully numeric matrix for modeling.

When looking at the new, cleaned table, I noticed a few helpful patterns. The classes are moderately imbalanced. About two thirds of outcomes are adoptions and one third are transfers. AgeAtOutcomeDays is right skewed, meaning many animals are very young which supports adoption rates. Dogs and cats dominate the shelter population, while other types appear less frequently. Most animals are already neutered by the time of their outcome. Animals with names also seem more likely to be adopted, which makes sense because names make animals feel more personable to potential adopters. These performers suggest that the engineered features carried useful signals for predicting Adoption vs Transfer.

To train the model I first started by framing the task as binary classification and used a stratified 80 to 20 train test split with a fixed random seed so the class ratios were preserved and the results were reproducible. All models shared the same scikit-learn pipeline. SimpleImputer with median strategy to handle the gaps, StandardScaler with mean set to False to normalize features without breaking sparse one hot matrices, and then the classifier. I trained a K Nearest Neighbors baseline with n_neighbors set to 11, weights set to distance, and p set to 2, plus a small memory safe GridSearchCV for KNN over k in {5, 11} and weights in {uniform, distance} with 3 fold cross validation and f1_macro scoring (12 fits total). I also trained Logistic Regression as a fast linear baseline, with a higher max to avoid convergence warnings. I collected and reported accuracy, precision, recall, and F1 for each class as well as confusion matrices. Since the classes are imbalanced, I decided to treat macro F1 as the most informative single summary. If the transfers were the operational priority, then recall for Transfer should become the key metric.

Overall, in terms of prediction, the KNN baseline achieved 0.8731 accuracy. For Adoption, precision was 0.8607, recall was 0.9556, and F1 was 0.9057. For Transfer, precision was 0.9031, recall was 0.7281, and F1 was 0.8062. The macro averages were listed as 0.8819 for precision, 0.8418 for recall, and 0.8559 for F1. The confusion pattern showed that most errors were true Transfers predicted as Adoptions, which does actually make some sense given overlapping signals and external factors that are not in the features. The GridSearchCV also confirmed that the baseline setting k = 11, weights = distance, and p = 2 was already the best in my small grid. Logistic Regression reached a very similar accuracy at about 0.8656, which suggests a largely linear separation.

In regards to the model, I am pretty confident that the pipeline is doing the right things and that the model has learned meaningful patterns. The dataset is quite large and the preparation steps are consistent. The test results are strong for a baseline approach leaning at about 87% accuracy with macro F1 around 0.856. The main limitation is Transfer recall, which is good but not as strong as Adoption recall, and this is expected since the transfer decisions depend on variables the model cannot predict.

In conclusion, this project took a lot of time but it was genuinely fun and I learned a lot! From cleaning and engineering features to building careful pipelines and interpreting real metrics like macro F1 and class specific recall, this project has taught me a lot. In the end, the models are accurate and explainable and I understand why they make the mistakes they do. Even though the runs sometimes took a while, the process was absolutely worth it because I now feel much more confident about taking a raw dataset to a well reasoned working classifier!