Jennifer Cheng
DATA71200 (Summer 2021): Advanced Data Analysis Methods
Professor Devaney
Final paper

Documenting a Machine Learning Approach to Predicting Penguin Species

To explore supervised and unsupervised algorithms in machine learning, I used a relatively straightforward dataset on the physical traits of penguins found in Antarctica. The following reflection of the preprocessing, modeling and evaluation process performed in projects 1-3 lays out the steps I took and what resulted. While I am not certain that I made the most sensible choices given my somewhat rudimentary comprehension of machine learning so far, I also attempt to identify what I might have done differently and what areas I intend to further explore to better my understanding of the algorithms and how to approach the machine learning procedure.

For the three projects, I used the Palmer penguins dataset, which in Python is available as part of the seaborn library. It contains variables concerning the physical characteristics of 344 penguins observed across three islands located on the Palmer Archipelago in Antarctica. In addition to their location, the features include the lengths of their bill and flipper, the depth of their bill, and their body mass and sex. The penguins belong to one of three species, the variable I used as the target or predictor attribute for the assignments. I chose the dataset because I wanted a prediction problem that would be clearly laid out – these physical characteristics and perhaps observed location of each penguin would be related to their species. And being new to machine learning, I didn't want too many variables, and potentially unclear ones, to complicate the task. I did wonder whether the number of samples would be too small.

To clean the data, I checked for missing values using `isnull()`; this returned eight observations that did not include a gender, with one of those rows missing all values except for the island on which the penguin was observed. I removed this one row altogether. To take care of the penguins with a missing gender, I imputed the values using the most frequent class (female). I

also checked that the target labels in both the training and test sets had the same proportions and distribution – Adelie penguins made up 44%, Gentoo 36% and Chinstrap 19.7% – by setting the `stratify=y` parameter in the `train_test_split()` function. Because the algorithms only take numeric data in scikit-learn, I then proceeded to transform the categorical data into numeric forms. In project 1, I used `LabelEncoder` from scikit-learn to encode the binary sex variable into 0 and 1. Upon reading that this kind of transformation of 0 and 1 should be used in cases of ordinal categorical data, I used one-hot encoding instead for the sex variable in project 2; I also applied the encoding to the other categorical variable of island, as well as the target attribute, which was the penguin species. This created new columns, one for each class label (i.e. each of the two genders, each of the three islands and each of the three species). If the particular gender, island or species applied to a penguin, that respective column would contain a 1 while the other columns for the sex, island or species would be 0. Given that the dataset was pretty straightforward without much ambiguity, I did not apply any form of feature selection or extraction; the seven variables all appeared to serve a purpose without creating noise or being redundant. In project 2, I attempted to scale the continuous numeric variables using scikit-learn's `StandardScaler` but wasn't sure at the time whether it was necessary or whether other scaling techniques would have been useful to implement, so I did not end up applying any form of scaling. Whether or not that was the right thing to do, I also chose not to standardize the data as the algorithms I used in project 2 did not require the numeric data to have a normal distribution. This would be something I would revisit along with explicitly checking for each column's data type to confirm, for example in this case, that the measurements were in fact numeric values.

In terms of visualizing the non-standardized numeric values, much of the data had more of a bimodal distribution, with two normal curves, when I plotted their histograms. If I were to recreate the charts, I would have color-coded the data by species and played with the number of

bins as that might have affected whether some of the physical traits would have taken on more of a normal distribution. When visualizing the data as a scatter matrix, I distinguished the samples using a different color for each species. This helped to identify, at least “superficially”, which penguins had longer bills and flippers, and weighed more, for example, as well as the characteristics shared by some of the species.

I then applied numeric transformations – in the form of squaring and cubing (good for linear regression models), and logarithmic and exponential (for getting the data into a Gaussian form) – to the variables for flipper length and bill length. For the former, which without any transformation took on a bimodal distribution, logarithmic and exponential applications showed the most change with more of the data tending toward the lower end of the flipper measurements. The bill length samples took on a relatively bell-shaped curve that became more right-skewed with the squaring and cubing transformations and a bit more bimodal with the logarithmic transformation. As was the case with the flipper lengths, the exponential transformation yielded a distribution that was primarily isolated to the first bin. This is an area I have to better understand, especially to see which kind of data this would work well for to visualize, such as in the case of count data, as mentioned in the Müller and Guido text (234).

From there, with the penguin dataset split into training and testing sets, cleaned and encoded, and establishing the species as the target attribute, I attempted to apply the k-nearest neighbors and decision tree (single tree) supervised learning algorithms. For each one, I ran the model with the default parameters using cross-validation, and then using grid search to compute favorable values for the hyperparameters.

The k-nearest neighbors (k-NN) method involves predicting the target label for test samples (in this case, the penguin species) by finding the nearest “k” (or “number of”) training data points (or “neighbors”). When instantiating the k-NN classifier, you can set k so that the

model will look for the k nearest neighbors (of the training set), then see how many of them belong to each of the different target classes. The class that contains the most of those k samples is the class assigned to the test data point in question (i.e. the predicted class). As noted in the “Supervised Learning with scikit-learn” Datacamp module, the k-NN algorithm basically establishes a set of decision boundaries between the classes. When using the default parameters with cross-validation, the accuracy score was 75%. I then used grid search to find the optimal values for k, weights (uniform or distance) and p (designating Manhattan, Euclidean, Minkowski distances), via `best_params_`. Inputting the resulting parameters (k=5, p=1 or Manhattan, weights="distance") from the search did indeed return a higher accuracy score of 82% in predicting the penguin species. Adjusting the number of neighbors to 9 and 4 to see what would happen, with the other parameters both changed and unchanged, led to lower accuracy scores (77.6%, 60%). As stated in class, k should be larger than and not be a multiple of the number of classes so perhaps that is one reason why setting `n_neighbors=5` contributed to a better performance than the other values. The other scores did, however, result in slightly higher weighted average precision, recall and F1 scores.

Decision trees, meanwhile, follow if-else questions, the answers of which determine the path down the hierarchy of nodes (“boxes”) and edges (connecting lines) – both of which correspond to the feature variables of the dataset – and reach a decision as to which class, or penguin species, the test observation belongs. (For example, if the body mass is greater than x grams, continue down this edge; otherwise continue down the other path.) This involves a recursive process that results in a binary tree of decisions based on true-false answers. When running a model using the decision tree classifier, the accuracy score that resulted from tuning the hyperparameters using grid search was much higher. Here, I set these parameters to tune the maximum tree depth (number of hierarchical layers), the minimum number of samples required

for a leaf node (the final decisions, or row of nodes), and the criterion function measuring a split quality (gini, entropy). Inputting the results of the grid search ('criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1) returned a 99% accuracy score, with precision, recall and F1 weighted average scores just hundredths of a point lower. Tweaking the parameters to higher values and "entropy" as the criterion resulted in slightly lower scores of 96.8% in training accuracy, and 91% in testing accuracy and precision, recall and F1 weighted average scores. Might this be because limiting the tree depth and number of leaves and splits helps to prevent overfitting and unnecessary nodes?

The better performance of the decision tree classifier could perhaps be the result of the recursive nature of the algorithm and the seemingly greater number of steps and decisions in determining the species class. All of the potential classes are considered and makes for a more comprehensive analysis of the dataset. (Which is not to say a decision tree will work better in all cases against a k-Nearest Neighbors classifier.)

In project 3, the use of principal component analysis (PCA), a decorrelation and dimension reduction technique, for feature selection introduced additional approaches to modeling the dataset via the supervised learning algorithms used in project 2. PCA disregards features with low variance and assumes that features with high variance are more informative. When fitting the PCA model on the penguins dataset, it determined that five out of nine features were needed for capturing 95% of the variation in the data. If I ran the grid search correctly on the decision tree model in project 2, the 99% accuracy score derived from a grid search would not be improved by running a PCA for feature selection. When running it in project 3, the accuracy score resulted in 100% on the training set and 98% on the test set, so comparable to the score in project 2 without PCA. Initially I found this concept a bit abstract and harder to comprehend following the

supervised learning chapter but the Datacamp modules helped break it down a bit especially in how the technique ties back to supervised learning.

PCA was then applied as a preprocessing step in the three clustering algorithms, which have overlapping but distinct traits. The k-means algorithm seeks to find cluster centroids that reflect regions of the data points. The algorithm goes back and forth between assigning points to the closest cluster center and establishing each center based on the mean of the data points assigned to the group. This recursive process results in changing cluster assignments but when samples are no longer being reassigned, the algorithm stops. In terms of downsides, k-means does not work well when clusters have more complex shapes and the algorithm uses random initialization (Müller and Guido 170-177).

Meanwhile, agglomerative (hierarchical) clustering involves a designating each data point as its own cluster and then merging the two clusters (or points) that are most alike – with three options for determining similarity – until a stopping criterion is reached; in the case of scikit-learn, that criterion would be a user-set number of clusters. This process can be visualized in the form of a dendrogram with all the classes at the bottom, each forming a cluster of one member; the two closest clusters, or classes, are merged thereby steadily decreasing the total number of clusters until just one remains that contains all the classes. Given this iterative process, agglomerative clustering does not make predictions for unseen data samples (184-188).

In DBSCAN, the clustering algorithm can work with clusters that form more complex shapes; it groups points together that are in dense regions composed of samples close together in space. Relatively empty spaces, in a way, draw the boundaries that separate the clusters, or dense regions of data. Unlike in the other algorithms, the user does not need to set the number of clusters before running the model; they can set the distance between points that would determine a neighborhood or cluster, thereby influencing cluster sizes. Points not clustered with

others, found in less dense regions, are considered noise, or outliers.

When running models with PCA for these clustering algorithms on the penguins dataset, it resulted in more densely-packed and perhaps even conspicuous clusters that are in closer proximities, a possible reflection of PCA's tendency for reducing noise and compressing data points; and the outcome of both data being rotated so that features are not statistically correlated and data that has been subset based on variance levels.

The three projects gave me a more tangible understanding of how the selected algorithms work. Though that understanding is far from comprehensive and I will continue to try to discern how the hyperparameters contribute to the performance of a model, running the models on a relatively uncomplicated dataset helped me to comprehend concepts that started out for me as rather abstract and difficult to differentiate. However, the projects showed that the machine learning process generally follows a checklist of steps to perform from data preparation, splitting, preprocessing to modeling and evaluation. I had some trouble with the proper sequence concerning splitting, cleaning and encoding the data but understanding that splitting the data happens prior to wrangling is key and the step-by-step layout of the assignments helped clarify that.

If I were to do the three projects over again, I would use a bit more complex dataset with more observations and more features and target classes. Reviewing the algorithms and the preprocessing and hyperparameter tuning concepts, I would try the other classification techniques (logistic regression, naive bayes, decision trees, support vectors machines) and compare the results and get a better understanding of how those supervised learning algorithms differ from each other. I would also try the different scaling methods to compare the results and see which would be more relevant for which kinds of data. Further, having a relatively improved grasp of unsupervised learning since project 3, I would try out more of the transformation

techniques; and also try to create more and better visualizations (as exploratory data analysis and from unsupervised learning techniques). I found visualizing through line graphs the accuracy results across a range of numeric values for hyperparameters helped quite a bit in providing some direction, especially when I did not know how to approach tuning the parameters, even when using grid search.

Overall, I hope to better my understanding of the supervised and unsupervised learning algorithms, how they can work together, and the machine learning process as a whole, and one day be able to revisit the projects and know where I may not have done something correctly and see more precisely what I could have done differently to improve the models.