HIGH SCHOOL EDITION
Journal of Student Research
Volume 11 Issue 4 (2022)

Using Machine Learning to Predict Penguin Species

Alex Hua[1] and Guillermo Goldsztein[#]

[1]Acton-Boxborough Regional High School, USA
[#]Advisor

ABSTRACT

Machine learning is used to make a neural network to predict penguin species from physical attributes. The data set is first prepared, with missing and extraneous data deleted. After this, there remains 333 examples consisting of three different penguin species. These remaining examples are fed into a neural network which proceeds to train on them. The neural network consists of an input and output layer with no hidden layers and uses the softmax activation formula. 25% of the data set is used as a validation set with the remaining 75% used as the training set.

Introduction

Machine learning is the use of data to develop computational models that make predictions and help inform decisions. The data used is a set of examples, with each example containing a set of features and a label. The list of all examples is called a data set. A model can be fed this data and trained on it in order to make predictions. The data set used in this paper consists of the physical characteristics of various penguins and what species they belong to. The features of a data set are the input that we give the model to train on (like the sex of a penguin). The label of a data set is the actual value of an example (with our data set of penguins, the label is what species each penguin belongs to). When our model makes a prediction, it should be the same as the label and can therefore be used to check the quality of our model. The goal of a machine learning model, in the case of the data set of the paper, would be to predict what species an individual penguin belongs to from only its physical characteristics. We refer the reader to the books [1, 2, 3] for more details on machine learning.

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE

Figure 1

The features in this data set are the island the penguin lives on (“Torgersen”, “Biscoe” and “Dream”), “culmen length”, “culmen depth”, “flipper length”, “body mass” and “sex”. The label is the penguin’s actual species (“Adelie”, “Chinstrap” and “Gentoo”). The labels in this particular data set are categorical, meaning they are not numbers and are categories like "yes" or "no". Additionally, there are more than two categories (the three species), so this study is considered a multi classification problem instead of a binary classification problem.

In order to create a model, a portion of the data set must be used to train the model. This is called the training set. The model will then train on the training set, constantly changing itself in order to minimize the error between the model’s prediction and the actual values.

Multi Classification Problem

Given that this is a multi-classification problem, we must first create a new label for each category. The original data set had one label called "species". Through a process called one hot encoding, three new labels will be created called "Adelie", "Chinstrap" and "Gentoo" to replace "species". During one hot encoding, the words Adelie, Chinstrap and Gentoo will be replaced by numbers, specifically ones and zeroes. When looking at one example, one penguin will be marked as "1" and the other two as "0". If the penguin in the example is an Adelie penguin, the example should look as follows:

Adelie	Chinstrap	Gentoo
1	0	0

Figure 2

Models of multi classification problems have multiple outputs corresponding to the number of labels. In the case of our data set, three species mean three outputs. The output of the model is usually written as $\hat{y}$ (Y hat). Thus, in our model, $\hat{y}_1$ = Adelie, $\hat{y}_2$ = Chinstrap and $\hat{y}_3$ = Gentoo. Our model will predict what species a penguin is by checking the probability of each category. For example, if $\hat{y}_1 = 0.7$, $\hat{y}_2 = 0.2$ and $\hat{y}_3 = 0.1$, this specific penguin would have a 70% chance of being Adelie, 20% chance of being Chinstrap and a 10% chance of being Gentoo. Since it has the highest chance of being an Adelie penguin, that is what the model will predict. Note, every probability will always be between 0 and 1 and the sum of all the probabilities will always equal 1.

Preparing the Data Set

Before we can start discussing the creation of a machine learning model, we have to prepare the data set. First off, we have to delete all examples in the data set that have missing or unneeded data. After searching the data set, there are examples missing data in “culmen length”, “culmen depth” and “sex”. All these examples can be deleted. Additionally, there is one example that just includes “.” as a feature in “sex”. This is clearly not a sex, and since there is only one example with ".", this example can also be deleted.

Second off, just like the penguin species, features that are categorical must be changed into numerical features. The only two features that are categorical are the island of each penguin and their sex. Since the island has three different categories just like the three penguin species, we can use the same process. If the penguin is from Torgersen, the example should like this:

Torgersen	Biscoe	Dream
1	0	0

Figure 3

When it comes to a penguin’s sex, it’s much easier, as there are only two categories. We can replace “male” with “0” and “female” with “1” without the need to create multiple variables.

MALE	0
FEMALE	1

Figure 4

Lastly, we have to check whether or not the data set is balanced; as in, are there a similar number of examples for each type of label? Checking, there are 146 examples with the label "Adelie", 68 with "Chinstrap" and 119 with "Gentoo". While there are clearly less examples of "Chinstrap", the gap is not enough to warrant any further action.

Overfitting and Validation Sets

When creating a model, one thing we must watch out for is overfitting our model to the training set. Overfitting occurs when we overcomplicate our model. A certain training set has a certain amount of noise in each label. Noise is the difference between a label and the predicted value our model will produce; it's the margin of error that real life has. Not every label will exactly match each prediction of the model. If we try to fit our model to a training set too much, it will capture too much of the noise of our training set. Then, if we try to give the model an example not in the training set, the model's prediction will be more inaccurate.

A way to avoid overfitting is a validation set. A validation set will be a portion of our original data set; in the case of the penguin data set, 25% will be used as a validation set, with the other 75% being the training set. While we try to minimize the error in the training set, the model will also be tested on the validation set. As long as the error continues to go down on both the training and validation sets, no overfitting is occurring. As soon as the error starts to go up on the validation set, we know the model has started to overfit on the training set.

Cross Entropy Error

We have mentioned how a model is made by trying to minimize the error, but what is that error? The error used in this model is cross entropy error, and while we will not get into specifics of it, the formula for a multi classification problem is:

$$J = \frac{-1}{n} \sum_{i=1}^{n} \quad \sum_{j=1}^{k} \quad y_{ij} \log(\hat{y}_{ij})$$

All we need to know is that when $\hat{y}$ (the predicted value) and y (the actual value) are closer together, the error is lower. When $\hat{y} = y$, the error is zero. Minimizing this is the whole point of the model.

Neural Networks

Considering that the penguin data set is a multi-classification problem, we are forced to use a neural network, as the other methods only work on numerical and binary classification problems. A neural network is made up of nodes and layers. All neural networks have at least two layers: an input layer and an output layer. The input layer takes in all the features of a data set. The output layer outputs the predicted values of the model. Since we have three labels - the three penguin species - our model will have three output nodes. We could also add hidden layers, layers in between the input and output layer, to increase accuracy if need be. Each layer has an activation function; some examples are the sigma, hyperbolic, relu, identity and softmax functions. Since no calculations happen in the input layer (as it just contains the data input), it has no activation function.

Scaling Our Features

Now that we understand each part of our model, we can apply that to our penguin data set. Earlier in this paper, we saw how the data set can be prepared by changing all categorical values to zeros and ones. Now we have to actually enter that data into a model. To enter the features, we first have to scale them.

With the variables of x_n as a feature, μ as the mean and σ as the standard deviation, the formula for scaling features is:

$$\frac{x_n - \mu}{\sigma}$$

The reason we scale features is to make them easier for the computer to use and learn with. Scaled features turn features into numbers that are not too small or large nor too clumped together or spread apart. These are ideal for the computer. An example of scaled features in this data set is:

```
array([[-0.13865906, -1.61882242,  0.48192924, ..., -0.39766762,
         1.00402416, -0.75891328],
       [ 0.45880371, -1.51367391,  0.62114082, ..., -0.39766762,
         1.00402416, -0.75891328],
       [ 1.21921087, -0.72506015,  1.73483345, ..., -0.39766762,
         1.00402416, -0.75891328],
       ...,
       [ 1.32784047,  1.00989013, -0.00531129, ..., -0.39766762,
        -0.99599197,  1.31767361],
       [ 0.42259385,  0.37899912, -0.70136918, ..., -0.39766762,
        -0.99599197,  1.31767361],
       [-0.91717116,  0.01097936, -0.70136918, ...,  2.51466288,
        -0.99599197, -0.75891328]])
```

Figure 5

Unlike the features, the labels in this data set do not have to be scaled, as they are categorical. If they were numerical, they would have to be scaled. Now that the features are in a state the computer can utilize, we can start building our neural network.

Fitting A Model to Our Data Set

Below is the first bit of code from the model:

```
model = 0
model = Sequential()
model.add(Dense(3, activation = 'softmax'))
```

Figure 6

The first line deletes any models that were created before. This is not entirely necessary but is just there as a cautionary measure so nothing messes with the model we intend to create. The second line creates the input layer of our neural network. The last line creates the output layer of our neural network. The “3” in the code specifies that there will be three outputs (the probabilities of each penguin species) and the activation function is the softmax function. For now, no hidden layers will be added. After testing our model for the first time, we can decide whether a hidden layer is necessary.

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.fit(X_train_scaled, y_train, epochs = 200, verbose = 0, validation_data = (X_val_scaled, y_val))
```

Figure 7

On to the next two lines of code. The first line in this image specifies the type of error (the cross-entropy error described earlier). The second part of the first line ("metrics = ['accuracy']") creates a table showing the accuracy of our model. Accuracy will be explained later. The second line starts entering in the data the model needs. The scaled features of our training set ("X_train_scaled") and the labels of our training set ("y_train") are inputted via the code at the beginning of the line, and the scaled features and labels of our validation set ("X_val_scaled" and "y_val") are inputted via the second part. We now have to specify the number of times we want our model to iterate; "epochs = 200" specifies that our model will iterate two hundred times. If our model is not good enough after two hundred epochs, we can increase that number. The "verbose = 0" just makes it so that a bunch of stuff does not appear on our screen as the model runs.

```
J_list = model.history.history['loss']
plt.plot(J_list)
J_list_val = model.history.history['val_loss']
plt.plot(J_list_val)
losses = pd.DataFrame(model.history.history)
losses
```

Figure 8

To track the progress of our model and to make sure it does not overfit, we need to be able to track the error of both the training and validation set. This is what the first four lines of code in this image do. The last two lines of code make a graph so that we can easily visually look at the error to see if our model has been successful. If the error on both the training and validation sets begins to flatten out and is low enough, our model is good enough.

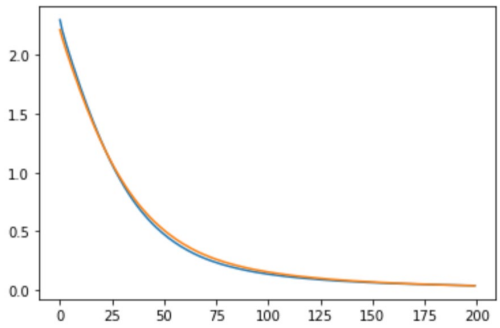

Figure 9

As we can see, after two hundred epochs the error on the training and validation sets (the blue and orange lines respectively) have settled to a relatively low error. Our model is complete, and our neural network does not need any hidden layers. Adding hidden layers just complicates things and can add to overfitting, so we want to keep our model as simple as we can.

Analyzing the Model

Now that our model is finished, how do we know if it's any good? There are three things that you can use to determine if the predictions of a model are good: accuracy, recall and precision. Accuracy is the number of correct predictions over the number of predictions. Basically, if you had one hundred you wanted to predict and got ninety-five correct predictions, your accuracy would be 95/100 or 95%. So, how good was the model we made in the previous section?

	loss	accuracy	val_loss	val_accuracy
0	2.294163	0.004016	2.209939	0.000000
1	2.211476	0.004016	2.145340	0.000000
2	2.147507	0.004016	2.087034	0.000000
3	2.088197	0.004016	2.032020	0.011905
4	2.031563	0.008032	1.978467	0.011905
...	...	...	...	...
195	0.041662	1.000000	0.040515	1.000000
196	0.041288	1.000000	0.040064	1.000000
197	0.040915	1.000000	0.039612	1.000000
198	0.040589	1.000000	0.039180	1.000000
199	0.040248	1.000000	0.038758	1.000000

Figure 10

This table shows the accuracy of the model. As you can see, the accuracy of our model on both the training set and validation set (column two and four respectively) is 100%.

When a data set is balanced, there is no need to look at anything besides the accuracy. The penguin data set used in this paper is balanced, so we do not actually have to do anything else. However, this paper will still explain what recall and precision are. Recall is the number of true positives out of the number of positives, and precision is the number of true positives out of the number of positive predictions. Looking at these two fractions is important if a data set is unbalanced. Imagine that we only had two penguin species in this data set, Adelie and Chinstrap. Now, if ninety-five of the penguins were Adelie and only five Chinstrap, if our model predicted all one hundred penguins were Adelie, the model would technically be 95% accurate. Obviously though, we can see that this is a terrible way to judge our model. Because there are significantly more Adelie penguins, the accuracy is still very high even if our model only spits out a single output. In this case, looking at the recall and precision would give a much better indicator to judge the quality of the model.

References

1. Müller, A., & Guido, S. (2016). *Introduction to Machine Learning with Python: A guide for data scientists*. O'Reilly Media

2. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media

3. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer Publishing