Confusion Matrix

A universal method you can use to deeply study how well any of your supervised learning predictors is doing is by using a confusion matrix.

You were introduced to a confusion matrix in the lab assignment dealing with the accuracy of the USPS post office reading hand-written zip code labels. A confusion matrix displays your model's predicted (testing set) outputs against the *true* observational values. This helps you see how well your algorithm was able to generalize and identify specific target labels, along with which labels were often confused. This can be helpful in increasing your accuracy, because you can then engineer additional features that help to better identify highly confusing targets, and then take another run through your data analysis pipeline.

Traditionally, the predicted targets are aligned on the X-axis of the matrix, and the true values are aligned on the Y-axis. Let's say you have the following data:

```
>>> import sklearn.metrics as metrics
>>> y_true = [1, 1, 2, 2, 3, 3]  # Actual, observed testing dataset valu
>>> y_pred = [1, 1, 1, 3, 2, 3]  # Predicted values from your model
```

The true labels are encoded data representing cats, dogs, and monkeys, for the three values. You can compute a confusion matrix using SciKit-Learn as follows:

```
>>> metrics.confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [1, 0, 1],
       [0, 1, 1]])
```

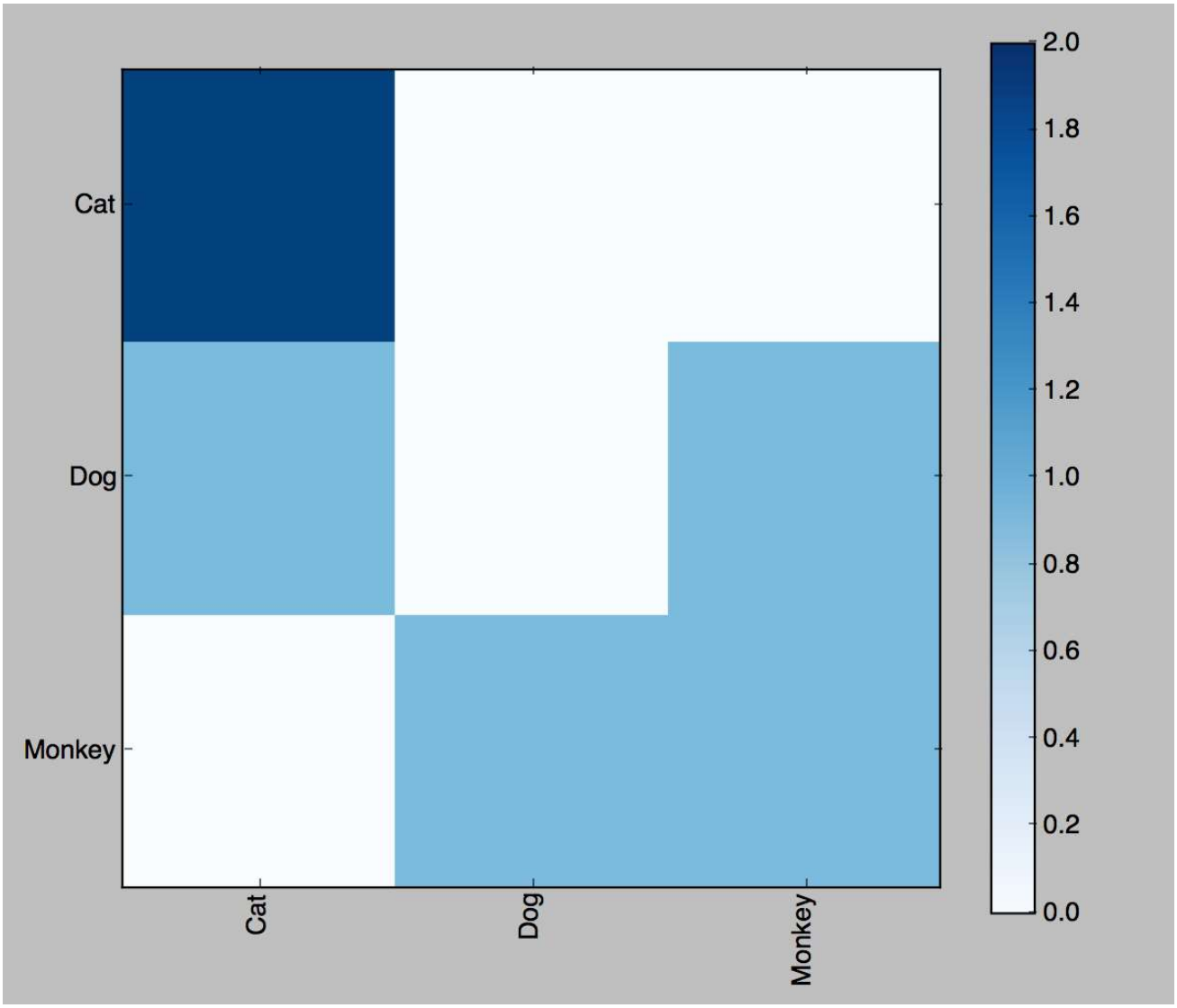
Perhaps a clearer representation of the data would look like this:

	Predicted Cat	Predicted Dog	Predicted Monkey
Actual Cat	2	0	0
Actual Dog	1	0	1
Actual Monkey	0	1	1

You can derive quite a bit of information from your confusion matrix. The first is how many actual cats, dogs, and monkeys you have. By summing up the values in a row, you'll know the true count of your data. You can do similarly with the columns, to see how many times your model predicted a certain target. By adding up all the values in the Predicted Dog column, we can see our model thought our testing dataset only had a single dog in it. An important thing to realize is that all of the non-diagonal elements of the matrix correspond to misclassified targets. Given all this information, you're able to derive probabilities relating to how accurate your answers are.

Given the example above, your algorithm predicted there were two cats in the dataset, and there indeed were two cats. In fact, the two samples the algorithm believed to be cats turned out to be the actual cats. It looks like the model is very good at identifying cats.

On the other hand, there were two dogs in the dataset. The model somehow came to the conclusion that one of the monkey's was a dog, and din't even arrive at two dog predictions. It looks like you trained a non-dog friendly model. Regarding monkeys, there were two in the dataset. One of them, the algorithm predicted correctly. The other, the model thought was a dog. It looks like there is some level of confusion here between monkeys and dogs. This is a good indicator that you might consider adding additional feature to your dataset, such as banana-affinity.



One last way you can look at this data is through using an `.imshow()` visualization plot:

```
>>> import matplotlib.pyplot as plt

>>> columns = ['Cat', 'Dog', 'Monkey']
>>> confusion = metrics.confusion_matrix(y_true, y_pred)

>>> plt.imshow(confusion, cmap=plt.cm.Blues, interpolation='nearest')
>>> plt.xticks([0,1,2], columns, rotation='vertical')
>>> plt.yticks([0,1,2], columns)
>>> plt.colorbar()

>>> plt.show()
```