**Essential Notes**

Page 9:

An important part of Scipy is scipy.parse, which provides us sparse matrices. It’s another representation of data used in scikit-learn.

from scipy import sparse

eye = np.eye(4)

print(“Numpy array:\n{}”.format(eye))

sparse\_matrix = sparse.csr\_mateix(eye)

print(“scipy sparse CSR matrix:\n{}”.format(sparse\_matrix))

SciPy sparse CSR matrix:

(0, 0) 1.0

(1, 1) 1.0

(2, 2) 1.0

(3, 3) 1.0

The reasoning behind this is that it is not usually possible to create “dense” representations of sparse data. This is because that data is ridiculously huge (in the orders of thousands of rows/columns in the matrix). We can use Scipy’s CSR format to get back the key values that are not spare and their locations. There’s also the COO method, in the book on page 9.

More on Scipy: scipy-lectures.org

Page 17:

We should not use our whole training set to fit the best model through the data. However, it is even more important to understand why. If we don’t save some data using sklearn’s train\_test\_split for a test set, the only way we will have to judge our model is to test it against itself (our predictions of the training data vs actual training data). The problem here is that the model is good at “remembering” the training set, since that it was it was trained against. We need to split, instead.

train\_test\_split uses a pseudo-random number generator to select which samples of the training set it should use for the test set, until it reaches the size it needs. This is incredibly important because otherwise, if it chose just the last test\_size% of the training set, it would only use the “2” label (virginica). That wouldn’t be ideal: we want diversification in both the training and test set. We specify random\_state to make the random selection process standard across my notebook and the textbook’s – it makes it deterministic.

Page 20:

When the features you are attempting to plot reach a dimension of 3D+, it is time to consider using a method called pair plotting to see the relationship between the features and the labels. Pair plotting allows all unique pairs of your features from your feature set to be compared on a 2D graph so that you can have an idea as to how your features stack up on the pair basis. While you still might be missing out on a full visualization, you still see quite a bit.

We use pandas.plotting.scatter\_matrix to create a pair plot.

from pandas.plotting import scatter\_matrix

scatter\_matrix(pd.Dataframe(data=X\_train, columns=dataset[“feature\_names”]), c=y\_train, marker=’o’, figsize=(15, 15), cmap=mglearn.cm3)

On the diagonal, there are the non-unique choices. There are 128 data points on each graph (75% of the original dataset was partitioned to the training set).

Page 21:

All scikit-learn models are implemented in their own called, called *Estimator* classes. Knn is implemented in sklearn.neighbors.KNeighborsClassifier.

Also mentions that when we create a KNeighborsClassifier object (and also provide my own paramaters), I get a modification-in-place of my object, with the parameters I passed applied and any other defaults, also applied. All of these other parameters are for special use cases or for speed optimizations.

Page 23:

To compare predictions of X\_test against y\_test, simply use np.mean() OR knn.score()

Method #1 🡪 np.mean()

np.mean(knn.predict(X\_test) == y\_test)

🡪 0.97934327894

Method #2 🡪 knn.score()

Pretty good!