**2. Import Classes and Functions**

We can begin by importing all of the classes and functions we will need in this tutorial.

This includes both the functionality we require from Keras, but also data loading from [pandas](http://pandas.pydata.org/) as well as data preparation and model evaluation from [scikit-learn](http://scikit-learn.org/).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import numpy  import pandas  from keras.models import Sequential  from keras.layers import Dense  from keras.wrappers.scikit\_learn import KerasClassifier  from keras.utils import np\_utils  from sklearn.model\_selection import cross\_val\_score  from sklearn.model\_selection import KFold  from sklearn.preprocessing import LabelEncoder  from sklearn.pipeline import Pipeline |

**3. Initialize Random Number Generator**

Next, we need to initialize the random number generator to a constant value (7).

This is important to ensure that the results we achieve from this model can be achieved again precisely. It ensures that the stochastic process of training a neural network model can be reproduced.



|  |  |
| --- | --- |
| 1  2  3 | # fix random seed for reproducibility  seed = 7  numpy.random.seed(seed) |

**4. Load The Dataset**

The dataset can be loaded directly. Because the output variable contains strings, it is easiest to load the data using pandas. We can then split the attributes (columns) into input variables (X) and output variables (Y).



|  |  |
| --- | --- |
| 1  2  3  4  5 | # load dataset  dataframe = pandas.read\_csv("iris.csv", header=None)  dataset = dataframe.values  X = dataset[:,0:4].astype(float)  Y = dataset[:,4] |

**5. Encode The Output Variable**

The output variable contains three different string values.

When modeling multi-class classification problems using neural networks, it is good practice to reshape the output attribute from a vector that contains values for each class value to be a matrix with a boolean for each class value and whether or not a given instance has that class value or not.

This is called [one hot encoding](https://en.wikipedia.org/wiki/One-hot) or creating dummy variables from a categorical variable.

For example, in this problem three class values are Iris-setosa, Iris-versicolor and Iris-virginica. If we had the observations:



|  |  |
| --- | --- |
| 1  2  3 | Iris-setosa  Iris-versicolor  Iris-virginica |

We can turn this into a one-hot encoded binary matrix for each data instance that would look as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | Iris-setosa, Iris-versicolor, Iris-virginica  1, 0, 0  0, 1, 0  0, 0, 1 |

We can do this by first encoding the strings consistently to integers using the scikit-learn class LabelEncoder. Then convert the vector of integers to a one hot encoding using the Keras function to\_categorical().



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # encode class values as integers  encoder = LabelEncoder()  encoder.fit(Y)  encoded\_Y = encoder.transform(Y)  # convert integers to dummy variables (i.e. one hot encoded)  dummy\_y = np\_utils.to\_categorical(encoded\_Y) |

**6. Define The Neural Network Model**

The Keras library provides wrapper classes to allow you to use neural network models developed with Keras in scikit-learn.

There is a KerasClassifier class in Keras that can be used as an Estimator in scikit-learn, the base type of model in the library. The KerasClassifier takes the name of a function as an argument. This function must return the constructed neural network model, ready for training.

Below is a function that will create a baseline neural network for the iris classification problem. It creates a simple fully connected network with one hidden layer that contains 8 neurons.

The hidden layer uses a rectifier activation function which is a good practice. Because we used a one-hot encoding for our iris dataset, the output layer must create 3 output values, one for each class. The output value with the largest value will be taken as the class predicted by the model.

The network topology of this simple one-layer neural network can be summarized as:



|  |  |
| --- | --- |
| 1 | 4 inputs -> [8 hidden nodes] -> 3 outputs |

Note that we use a “*softmax*” activation function in the output layer. This is to ensure the output values are in the range of 0 and 1 and may be used as predicted probabilities.

Finally, the network uses the efficient Adam gradient descent optimization algorithm with a logarithmic loss function, which is called “*categorical\_crossentropy*” in Keras.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # define baseline model  def baseline\_model():  # create model  model = Sequential()  model.add(Dense(8, input\_dim=4, activation='relu'))  model.add(Dense(3, activation='softmax'))  # Compile model  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  return model |

We can now create our KerasClassifier for use in scikit-learn.

We can also pass arguments in the construction of the KerasClassifier class that will be passed on to the fit() function internally used to train the neural network. Here, we pass the number of epochs as 200 and batch size as 5 to use when training the model. Debugging is also turned off when training by setting verbose to 0.



|  |  |
| --- | --- |
| 1 | estimator = KerasClassifier(build\_fn=baseline\_model, epochs=200, batch\_size=5, verbose=0) |

**7. Evaluate The Model with k-Fold Cross Validation**

We can now evaluate the neural network model on our training data.

The scikit-learn has excellent capability to evaluate models using a suite of techniques. The gold standard for evaluating machine learning models is k-fold cross validation.

First we can define the model evaluation procedure. Here, we set the number of folds to be 10 (an excellent default) and to shuffle the data before partitioning it.



|  |  |
| --- | --- |
| 1 | kfold = KFold(n\_splits=10, shuffle=True, random\_state=seed) |

Now we can evaluate our model (estimator) on our dataset (X and dummy\_y) using a 10-fold cross-validation procedure (kfold).

Evaluating the model only takes approximately 10 seconds and returns an object that describes the evaluation of the 10 constructed models for each of the splits of the dataset.



|  |  |
| --- | --- |
| 1  2 | results = cross\_val\_score(estimator, X, dummy\_y, cv=kfold)  print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100)) |

The results are summarized as both the mean and standard deviation of the model accuracy on the dataset. This is a reasonable estimation of the performance of the model on unseen data. It is also within the realm of known top results for this problem.



|  |  |
| --- | --- |
| 1 | Accuracy: 97.33% (4.42%) |

**Summary**

In this post you discovered how to develop and evaluate a neural network using the Keras Python library for deep learning.

By completing this tutorial, you learned:

* How to load data and make it available to Keras.
* How to prepare multi-class classification data for modeling using one hot encoding.
* How to use Keras neural network models with scikit-learn.
* How to define a neural network using Keras for multi-class classification.
* How to evaluate a Keras neural network model using scikit-learn with k-fold cross validation

Do you have any questions about deep learning with Keras or this post?

Ask your questions in the comments below and I will do my best to answer them.