Devansh Gupta dxg397 PS7

We can begin by importing all of the classes and functions we will need. This includes both the functionality we require from Keras, but also data loading from pandas as well as data preparation and model evaluation from scikit-learn.

```
import numpy
import pandas as pd
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
```

Encode The Output Variable

The target DataFrame contains 464 different string values.

```
# Load data
inputs = pd.read_csv('basf_inputs_rand_normalized.csv', sep=",")
targets = pd.read_csv ('basf_targets_rand_normalized.csv', sep=",")
```

```
targets.shape
```

```
(1028, 1)
```

```
targets.drop_duplicates().shape
```

```
(464, 1)
```

When modeling multi-class classification problems using neural networks, it is good practice to encode the data into One-Hot encoded data i.e. 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.

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().

```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(targets)
encoded_Y = encoder.transform(targets)
# convert integers to dummy variables (i.e. one hot encoded)
targets = np_utils.to_categorical(encoded_Y)
targets = pd.DataFrame(targets)
```

Define The Neural Network Model

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. It creates a simple fully connected network with one hidden layer that contains 20 neurons.

The hidden layer uses a rectifier activation function which is a good practice. Because we used a one-hot encoding for our dataset, the output layer must create 464 output values, one for each class. The output value with the largest value will be taken as the class predicted by the model.

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 two different loss functions, "*categorical_crossentropy*" and "mean_squared_error" in Keras.Below are the graphs output by the classifier with the two loss functions. The best accuracy is found using the categorial_crossentropy loss function.

```
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=12, activation='relu'))
    model.add(Dense(464, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100, verbose=0)
```

```
history = estimator.fit(data_train_x, data_train_y, validation_data=(data_val_x, data_val_y), validation_steps=10,
                        steps_per_epoch=50, epochs=100)
```

Figure : Baseline_model

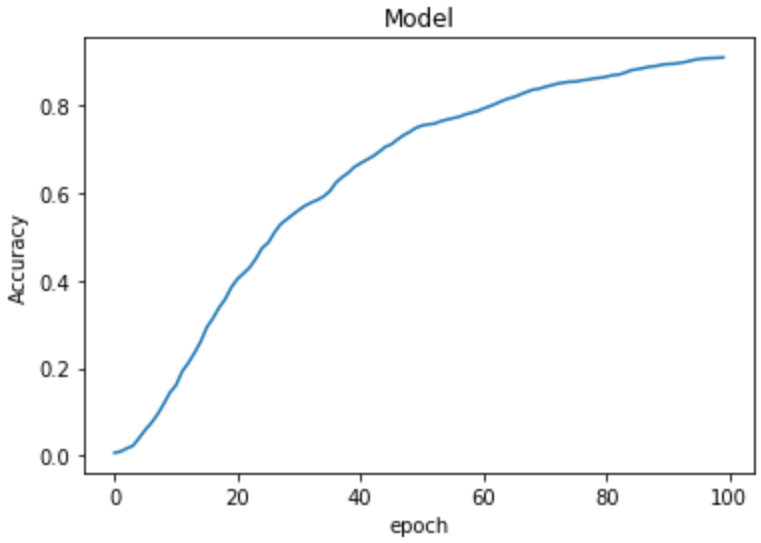

Figure: Accuracy of the classifier with loss function as categorial_crossentropy

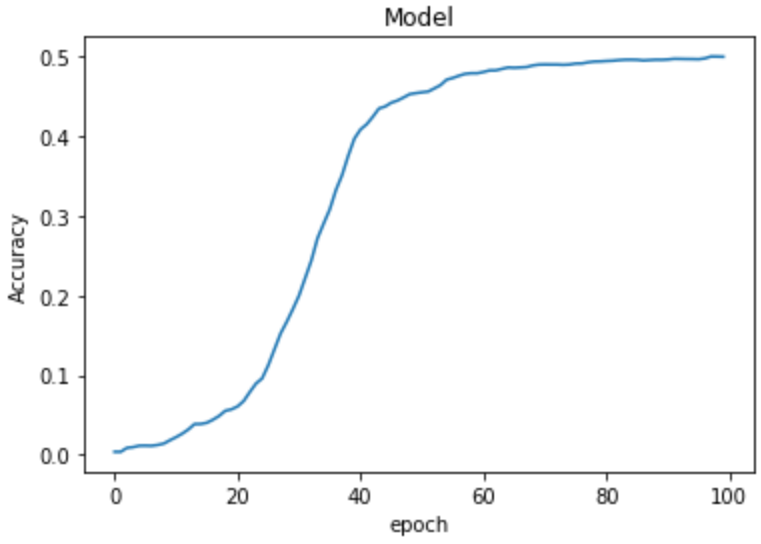

Figure : Accuracy of the classifier with loss function as mean_squared_error