MNIST Neural Shrub - Classes

July 15, 2019

```
In [1]: import time
        import mnist
        import numpy as np

        from sklearn.tree import DecisionTreeClassifier

        from keras.utils import np_utils
        from keras import Sequential
        from keras import layers
```

```
/home/shashwati/anaconda3/envs/py35/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWarn
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
In [2]: # gets the data
        def get_data():
            test = mnist.test_images()
            label_test = mnist.test_labels()
            train = mnist.train_images()
            label_train = mnist.train_labels()

            nsamples, nx, ny = train.shape
            train = train.reshape((nsamples,nx*ny))

            nsamples, nx, ny = test.shape
            test = test.reshape((nsamples,nx*ny))

            return train, label_train, test, label_test
```

```
In [3]: from sklearn.tree._tree import TREE_LEAF

        def prune_index(inner_tree, index, threshold):
            if inner_tree.value[index].min() < threshold:
                # turn node into a leaf by "unlinking" its children
                inner_tree.children_left[index] = TREE_LEAF
                inner_tree.children_right[index] = TREE_LEAF
            # if there are shildren, visit them as well
            if inner_tree.children_left[index] != TREE_LEAF:
                prune_index(inner_tree, inner_tree.children_left[index], threshold)
                prune_index(inner_tree, inner_tree.children_right[index], threshold)

In [4]: # builds the decision tree of depth 10
        def decision_tree(train, label):
            dt = DecisionTreeClassifier(max_depth = 10, random_state = 1)
            dt.fit(train, label)
            prune_index(dt.tree_, 0, 1)
            return dt

In [5]: # building the neural network
        def neural_network(class_data):
            num_train = []
            num_label = []
            for x in class_data:
                num_train.append(x[0])
                num_label.append(x[1])

            n_classes = 10
            num_train = np.matrix(num_train).astype('float32')/255
            num_label = np.array(num_label)
            num_label = np_utils.to_categorical(num_label, n_classes)

            model = Sequential()
            model.add(layers.Dense(512, activation = 'relu', input_shape=(784,)))
            model.add(layers.Dense(512, activation = 'relu'))
            model.add(layers.Dense(10, activation = 'softmax'))

            model.compile(loss='categorical_crossentropy', metrics=['accuracy'], \
                          optimizer='adam')
            model.fit(num_train, num_label, batch_size=128, epochs=10)
            return model

In [6]: # building the neural network for each class
        def neural_shrubs(tree, train, label):
            train = np.array(train)
            label = np.array(label)

            # leave_id: index of the leaf that cantains the instance
            leave_id = tree.apply(train)

            num_class = 10
            classes = [[] for i in range(0, num_class)]

            for x in range(len(train)):
                leaf = leave_id[x]
                # Gets the class for each leaf
                #.value: returns the distributition at the leaf,
                #        i.e number of instance in each class at that leaf
                #.argmax(): returns the class which has the max instance
                #        i.e here: (0, 1, 2) - it is 0-indexed
                idx = np.array(tree.tree_.value[leaf]).argmax()

                # insert the instance into the class
                classes[idx].append([train[x], label[x]])

            # stores the neural network for each class
            nn_models = []

            #stores the max time taken to build a neural network
            max_time = 0;

            for x in range(num_class):
                start = time.time()
                model = neural_network(classes[x])
                end = time.time()

                time_taken = end - start
                if max_time < time_taken:
                    max_time = time_taken

                nn_models.append(model)

            # returns a neural network for each class and the max
            # time taken to build the neural network
            return nn_models, max_time
```

```
In [7]: # The algorithm to build the neural shrub
        train, train_label, test, test_label = get_data()

        dt_start = time.time()
        tree = decision_tree(train, train_label)
        dt_end = time.time()

        shrubs, max_time = neural_shrubs(tree, train, train_label)
```

```
Epoch 1/10
5440/5440 [==============================] - 1s 215us/step - loss: 0.4098 - acc: 0.8866
Epoch 2/10
5440/5440 [==============================] - 1s 134us/step - loss: 0.1360 - acc: 0.9603
Epoch 3/10
5440/5440 [==============================] - 1s 136us/step - loss: 0.0756 - acc: 0.9776
Epoch 4/10
```

```
5440/5440 [==============================] - 1s 139us/step - loss: 0.0472 - acc: 0.9860
Epoch 5/10
5440/5440 [==============================] - 1s 137us/step - loss: 0.0336 - acc: 0.9901
Epoch 6/10
5440/5440 [==============================] - 1s 132us/step - loss: 0.0187 - acc: 0.9960
Epoch 7/10
5440/5440 [==============================] - 1s 141us/step - loss: 0.0083 - acc: 0.9983
Epoch 8/10
5440/5440 [==============================] - 1s 139us/step - loss: 0.0040 - acc: 0.9996
Epoch 9/10
5440/5440 [==============================] - 1s 139us/step - loss: 0.0020 - acc: 1.0000
Epoch 10/10
5440/5440 [==============================] - 1s 137us/step - loss: 0.0012 - acc: 1.0000
Epoch 1/10
7303/7303 [==============================] - 1s 188us/step - loss: 0.3888 - acc: 0.8970
Epoch 2/10
7303/7303 [==============================] - 1s 138us/step - loss: 0.1208 - acc: 0.9634
Epoch 3/10
7303/7303 [==============================] - 1s 139us/step - loss: 0.0662 - acc: 0.9817
Epoch 4/10
7303/7303 [==============================] - 1s 135us/step - loss: 0.0408 - acc: 0.9881
Epoch 5/10
7303/7303 [==============================] - 1s 139us/step - loss: 0.0317 - acc: 0.9901
Epoch 6/10
7303/7303 [==============================] - 1s 135us/step - loss: 0.0130 - acc: 0.9971
Epoch 7/10
7303/7303 [==============================] - 1s 137us/step - loss: 0.0079 - acc: 0.9988
Epoch 8/10
7303/7303 [==============================] - 1s 149us/step - loss: 0.0084 - acc: 0.9971
Epoch 9/10
7303/7303 [==============================] - 1s 143us/step - loss: 0.0057 - acc: 0.9989
Epoch 10/10
7303/7303 [==============================] - 1s 134us/step - loss: 0.0030 - acc: 0.9996
Epoch 1/10
5625/5625 [==============================] - 1s 219us/step - loss: 0.5039 - acc: 0.8583
Epoch 2/10
5625/5625 [==============================] - 1s 154us/step - loss: 0.1965 - acc: 0.9415
Epoch 3/10
5625/5625 [==============================] - 1s 155us/step - loss: 0.1121 - acc: 0.9696
Epoch 4/10
5625/5625 [==============================] - 1s 148us/step - loss: 0.0693 - acc: 0.9829
Epoch 5/10
5625/5625 [==============================] - 1s 140us/step - loss: 0.0415 - acc: 0.9874
Epoch 6/10
5625/5625 [==============================] - 1s 138us/step - loss: 0.0214 - acc: 0.9954
Epoch 7/10
5625/5625 [==============================] - 1s 136us/step - loss: 0.0154 - acc: 0.9973
Epoch 8/10
```

```
5625/5625 [==============================] - 1s 135us/step - loss: 0.0095 - acc: 0.9980
Epoch 9/10
5625/5625 [==============================] - 1s 136us/step - loss: 0.0041 - acc: 1.0000
Epoch 10/10
5625/5625 [==============================] - 1s 138us/step - loss: 0.0020 - acc: 1.0000
Epoch 1/10
4914/4914 [==============================] - 1s 217us/step - loss: 0.5001 - acc: 0.8541
Epoch 2/10
4914/4914 [==============================] - 1s 133us/step - loss: 0.1905 - acc: 0.9394
Epoch 3/10
4914/4914 [==============================] - 1s 136us/step - loss: 0.1057 - acc: 0.9670
Epoch 4/10
4914/4914 [==============================] - 1s 136us/step - loss: 0.0615 - acc: 0.9807
Epoch 5/10
4914/4914 [==============================] - 1s 141us/step - loss: 0.0454 - acc: 0.9878
Epoch 6/10
4914/4914 [==============================] - 1s 138us/step - loss: 0.0296 - acc: 0.9912
Epoch 7/10
4914/4914 [==============================] - 1s 135us/step - loss: 0.0142 - acc: 0.9974
Epoch 8/10
4914/4914 [==============================] - 1s 134us/step - loss: 0.0073 - acc: 0.9994
Epoch 9/10
4914/4914 [==============================] - 1s 140us/step - loss: 0.0041 - acc: 0.9998
Epoch 10/10
4914/4914 [==============================] - 1s 130us/step - loss: 0.0026 - acc: 0.9998
Epoch 1/10
4926/4926 [==============================] - 1s 224us/step - loss: 0.5462 - acc: 0.8601
Epoch 2/10
4926/4926 [==============================] - 1s 139us/step - loss: 0.2018 - acc: 0.9397
Epoch 3/10
4926/4926 [==============================] - 1s 138us/step - loss: 0.1091 - acc: 0.9679
Epoch 4/10
4926/4926 [==============================] - 1s 131us/step - loss: 0.0610 - acc: 0.9827
Epoch 5/10
4926/4926 [==============================] - 1s 136us/step - loss: 0.0368 - acc: 0.9905
Epoch 6/10
4926/4926 [==============================] - 1s 145us/step - loss: 0.0209 - acc: 0.9953
Epoch 7/10
4926/4926 [==============================] - 1s 135us/step - loss: 0.0135 - acc: 0.9974
Epoch 8/10
4926/4926 [==============================] - 1s 142us/step - loss: 0.0066 - acc: 0.9990
Epoch 9/10
4926/4926 [==============================] - 1s 142us/step - loss: 0.0051 - acc: 0.9990
Epoch 10/10
4926/4926 [==============================] - 1s 141us/step - loss: 0.0049 - acc: 0.9992
Epoch 1/10
5640/5640 [==============================] - 1s 226us/step - loss: 0.7148 - acc: 0.7752
Epoch 2/10
```

```
5640/5640 [==============================] - 1s 168us/step - loss: 0.2608 - acc: 0.9158
Epoch 3/10
5640/5640 [==============================] - 1s 153us/step - loss: 0.1466 - acc: 0.9544
Epoch 4/10
5640/5640 [==============================] - 1s 146us/step - loss: 0.1241 - acc: 0.9585
Epoch 5/10
5640/5640 [==============================] - 1s 138us/step - loss: 0.0607 - acc: 0.9835
Epoch 6/10
5640/5640 [==============================] - 1s 140us/step - loss: 0.0427 - acc: 0.9906
Epoch 7/10
5640/5640 [==============================] - 1s 134us/step - loss: 0.0239 - acc: 0.9957
Epoch 8/10
5640/5640 [==============================] - 1s 139us/step - loss: 0.0128 - acc: 0.9991
Epoch 9/10
5640/5640 [==============================] - 1s 143us/step - loss: 0.0097 - acc: 0.9991
Epoch 10/10
5640/5640 [==============================] - 1s 146us/step - loss: 0.0055 - acc: 1.0000
Epoch 1/10
6362/6362 [==============================] - 1s 215us/step - loss: 0.4952 - acc: 0.8544
Epoch 2/10
6362/6362 [==============================] - 1s 130us/step - loss: 0.1732 - acc: 0.9491
Epoch 3/10
6362/6362 [==============================] - 1s 131us/step - loss: 0.1005 - acc: 0.9719
Epoch 4/10
6362/6362 [==============================] - 1s 128us/step - loss: 0.0627 - acc: 0.9827
Epoch 5/10
6362/6362 [==============================] - 1s 132us/step - loss: 0.0416 - acc: 0.9881
Epoch 6/10
6362/6362 [==============================] - 1s 208us/step - loss: 0.0221 - acc: 0.9937
Epoch 7/10
6362/6362 [==============================] - 2s 252us/step - loss: 0.0142 - acc: 0.9978
Epoch 8/10
6362/6362 [==============================] - 1s 162us/step - loss: 0.0085 - acc: 0.9984
Epoch 9/10
6362/6362 [==============================] - 1s 152us/step - loss: 0.0069 - acc: 0.9989
Epoch 10/10
6362/6362 [==============================] - 1s 126us/step - loss: 0.0028 - acc: 1.0000 0s - lo
Epoch 1/10
6107/6107 [==============================] - 1s 215us/step - loss: 0.4186 - acc: 0.8832
Epoch 2/10
6107/6107 [==============================] - 1s 134us/step - loss: 0.1486 - acc: 0.9556
Epoch 3/10
6107/6107 [==============================] - 1s 199us/step - loss: 0.0872 - acc: 0.9740
Epoch 4/10
6107/6107 [==============================] - 1s 187us/step - loss: 0.0533 - acc: 0.9849
Epoch 5/10
6107/6107 [==============================] - 1s 137us/step - loss: 0.0351 - acc: 0.9892
Epoch 6/10
```

```
6107/6107 [==============================] - 1s 129us/step - loss: 0.0232 - acc: 0.9936
Epoch 7/10
6107/6107 [==============================] - 1s 131us/step - loss: 0.0117 - acc: 0.9979
Epoch 8/10
6107/6107 [==============================] - 1s 133us/step - loss: 0.0058 - acc: 0.9993
Epoch 9/10
6107/6107 [==============================] - 1s 123us/step - loss: 0.0072 - acc: 0.9982
Epoch 10/10
6107/6107 [==============================] - 1s 138us/step - loss: 0.0055 - acc: 0.9984
Epoch 1/10
6578/6578 [==============================] - 1s 211us/step - loss: 0.7098 - acc: 0.7806
Epoch 2/10
6578/6578 [==============================] - 1s 126us/step - loss: 0.2681 - acc: 0.9220
Epoch 3/10
6578/6578 [==============================] - 1s 129us/step - loss: 0.1813 - acc: 0.9465
Epoch 4/10
6578/6578 [==============================] - 1s 134us/step - loss: 0.1229 - acc: 0.9641
Epoch 5/10
6578/6578 [==============================] - 1s 131us/step - loss: 0.0837 - acc: 0.9758
Epoch 6/10
6578/6578 [==============================] - 1s 129us/step - loss: 0.0569 - acc: 0.9839
Epoch 7/10
6578/6578 [==============================] - 1s 127us/step - loss: 0.0408 - acc: 0.9906
Epoch 8/10
6578/6578 [==============================] - 1s 124us/step - loss: 0.0208 - acc: 0.9964
Epoch 9/10
6578/6578 [==============================] - 1s 127us/step - loss: 0.0137 - acc: 0.9977
Epoch 10/10
6578/6578 [==============================] - 1s 129us/step - loss: 0.0148 - acc: 0.9968
Epoch 1/10
7105/7105 [==============================] - 1s 206us/step - loss: 0.6220 - acc: 0.8083
Epoch 2/10
7105/7105 [==============================] - 1s 127us/step - loss: 0.2009 - acc: 0.9417
Epoch 3/10
7105/7105 [==============================] - 1s 129us/step - loss: 0.1099 - acc: 0.9707
Epoch 4/10
7105/7105 [==============================] - 1s 136us/step - loss: 0.0696 - acc: 0.9807
Epoch 5/10
7105/7105 [==============================] - 1s 140us/step - loss: 0.0422 - acc: 0.9896
Epoch 6/10
7105/7105 [==============================] - 1s 126us/step - loss: 0.0270 - acc: 0.9928
Epoch 7/10
7105/7105 [==============================] - 1s 129us/step - loss: 0.0145 - acc: 0.9973
Epoch 8/10
7105/7105 [==============================] - 1s 129us/step - loss: 0.0127 - acc: 0.9975
Epoch 9/10
7105/7105 [==============================] - 1s 125us/step - loss: 0.0062 - acc: 0.9997
Epoch 10/10
```

```
7105/7105 [==============================] - 1s 134us/step - loss: 0.0045 - acc: 0.9993
```

```
In [8]: # predicts using the neural shrub
        def neural_shrub_predict(tree, nn_model, test, label):
            label_test = np.array(label)
            test = np.array(test)

            #row - actual; col - pred
            confusion_matrix = np.zeros((10,10), dtype=np.int)
            correct = 0

            for i in range(len(test)):
                x = test[i]
                pred_class = tree.predict([x])
                x = np.array([x])
                nn_model_class = nn_model[pred_class[0]]
                pred = np.argmax(nn_model_class.predict(x))

                confusion_matrix[label[i]][pred] = \
                    confusion_matrix[label[i]][pred] + 1
                if pred == label[i]: correct = correct + 1

            acc_score = correct/len(test)

            return confusion_matrix, acc_score
```

```
In [9]: # function to calculate the metrics
        def metrics(cm, cls, size):
            cm = np.array(cm)
            tp = cm[cls][cls]
            fp = sum(cm[x, cls] for x in range(10))-cm[cls][cls]
            fn = sum(cm[cls, x] for x in range(10))-cm[cls][cls]
            tn = size - tp - fp - fn
            precision = tp/(tp+fp)
            recall = tp/(tp+fn)
            fmeasure = 2*(precision*recall)/(precision + recall)
            accuracy = (tp + tn)/size

            return precision, recall, fmeasure, accuracy
```

```
In [10]: # Predicting
         cm, acc_score = neural_shrub_predict(tree, shrubs, test, test_label)
         print("Confusion Matrix:\n\n", cm)
```

```
Confusion Matrix:

 [[ 970    0    1    2    0    2    2    1    1    1]
 [    0 1123    2    3    0    2    1    2    2    0]
 [   4    2  988   15    5    1    3    5    7    2]
 [   1    0    7  978    1    6    0    5    6    6]
 [   1    1    0    0  944    0    5    6    3   22]
 [   3    1    0   20    2  841    8    0   11    6]
 [   6    3    1    1    4    5  935    0    3    0]
 [   2    5    9    4    5    1    0  993    1    8]
 [   4    3    5   10    6    9    3    5  922    7]
 [   2    3    2    6   14    3    1   10    3  965]]
```

```
In [11]: # Class 0
         precision0, recall0, f0, acc0 = metrics(cm, 0, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 0: ", round(precision0, 3), "  ", round(recall0, 3), \
               " ", round(f0, 3), "   ", round(acc0,3))
```

```
        Precision Recall F-measure Accuracy
Class 0:  0.977    0.99   0.983     0.997
```

```
In [12]: # Class 1
         precision1, recall1, f1, acc1 = metrics(cm, 1, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 1: ", round(precision1, 3), "  ", round(recall1, 3), \
               " ", round(f1, 3), "   ", round(acc1,3))
```

```
        Precision Recall F-measure Accuracy
Class 1:  0.984    0.989   0.987     0.997
```

```
In [13]: # Class 2
         precision2, recall2, f2, acc2 = metrics(cm, 2, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 2: ", round(precision2, 3), "  ", round(recall2, 3), \
               " ", round(f2, 3), "   ", round(acc2,3))
```

```
        Precision Recall F-measure Accuracy
Class 2:  0.973    0.957   0.965     0.993
```

```
In [14]: # Class 3
         precision3, recall3, f3, acc3 = metrics(cm, 3, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 3: ", round(precision3, 3), "  ", round(recall3, 3), \
               " ", round(f3, 3), "   ", round(acc3,3))
```

```
        Precision Recall F-measure Accuracy
Class 3:  0.941    0.968   0.955     0.991
```

```
In [15]: # Class 4
         precision4, recall4, f4, acc4 = metrics(cm, 4, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 4: ", round(precision4, 3), "  ", round(recall4, 3), \
               " ", round(f4, 3), "   ", round(acc4,3))
```

```
        Precision Recall F-measure Accuracy
Class 4:  0.962    0.961   0.962     0.992
```

```
In [16]: # Class 5
         precision5, recall5, f5, acc5 = metrics(cm, 5, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 5: ", round(precision5, 3), "  ", round(recall5, 3), \
               " ", round(f5, 3), "   ", round(acc5,3))
```

```
        Precision Recall F-measure Accuracy
Class 5:  0.967    0.943   0.955     0.992
```

```
In [17]: # Class 5
         precision5, recall5, f5, acc5 = metrics(cm, 5, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 5: ", round(precision5, 3), "  ", round(recall5, 3), \
               " ", round(f5, 3), "   ", round(acc5,3))
```

```
        Precision Recall F-measure Accuracy
Class 5:  0.967    0.943   0.955     0.992
```

```
In [18]: # Class 6
         precision6, recall6, f6, acc6 = metrics(cm, 6, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 6: ", round(precision6, 3), "  ", round(recall6, 3), \
               " ", round(f6, 3), "   ", round(acc6,3))
```

```
        Precision Recall F-measure Accuracy
Class 6:  0.976    0.976   0.976     0.995
```

```
In [19]: # Class 7
         precision7, recall7, f7, acc7 = metrics(cm, 7, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 7: ", round(precision0, 3), "  ", round(recall7, 3), \
               " ", round(f7, 3), "   ", round(acc7,3))
```

```
        Precision Recall F-measure Accuracy
Class 7:  0.977    0.966   0.966     0.993
```

```
In [20]: # Class 8
         precision8, recall8, f8, acc8 = metrics(cm, 8, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 8: ", round(precision8, 3), "  ", round(recall8, 3), \
               " ", round(f8, 3), "   ", round(acc8,3))
```

```
        Precision Recall F-measure Accuracy
Class 8:  0.961    0.947   0.954     0.991
```

```
In [21]: # Class 9
         precision9, recall9, f9, acc9 = metrics(cm, 9, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 9: ", round(precision9, 3), "  ", round(recall9, 3), \
               " ", round(f9, 3), "   ", round(acc9,3))
```

```
        Precision Recall F-measure Accuracy
Class 9:  0.949    0.956   0.953     0.99
```

```
In [22]: # number of instances classified correctly
         print("Accuracy_score: ", round(acc_score, 5))
```

```
Accuracy_score:  0.9659
```

```
In [23]: # training time
         total_time_taken = dt_end - dt_start + max_time
         print("Training Time: %s sec" % round(total_time_taken, 5))
```

```
Training Time: 38.74031 sec
```