Neural Shrub - Classes

July 10, 2019

1 Neural Shrub - Classes

```
In [1]: import time
        import os, sys
        import string

        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import accuracy_score

        from keras.utils import np_utils
        from sklearn.preprocessing import LabelEncoder
        from keras import Sequential
        from keras import layers

        import pandas as pd
        import numpy as np
```

```
/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]: def load_data(dataset):
            if os.path.isfile(dataset):
                print("Loading ", dataset, " dataset ...")
                data = pd.read_csv(dataset)
                print("\nDataset loaded successfully\n\n")
                return data
            else:
                print('File not found')
                print('\n\nExiting...')
                sys.exit()
```

```
In [3]: #The column names are [a, b, c, ..., z, A, B, C, ..., W]
        columnNames = list(string.ascii_lowercase) \
                    + list(string.ascii_uppercase)[:23]
```

```
In [4]: def get_data():
            train_dataset = load_data('./sensIT_train.csv')
            label_train = train_dataset['result']
            train = train_dataset[columnNames]

            test_dataset = load_data('./sensIT_test.csv')
            label_test = test_dataset['result']
            test = test_dataset[columnNames]

            return train, label_train, test, label_test
```

```
In [5]: 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 [6]: # Makes the decision tree
        def decision_tree(train, label):
            dt = DecisionTreeClassifier(max_depth = 8, min_samples_leaf=500, random_state = 1)
            dt.fit(train, label)
            prune_index(dt.tree_, 0, 5)
            return dt
```

```
In [7]: # Class_data: list of instances belonging to a class
        # Each instance consists of the predictor_values and the actual class
        def neural_network(class_data):
            nn_train = []
            nn_label = []

            for instance in class_data:
                nn_train.append(instance[0]) # predictor
                nn_label.append(instance[1]) # actual class

            nn_train = np.array(nn_train)
            nn_label = np.array(nn_label)

            # Preprocessing
            encoder = LabelEncoder()
            encoder.fit(nn_label)
            nn_label = encoder.transform(nn_label)
            nn_label = np_utils.to_categorical(nn_label)
```

```
    # Neural network structure
    model = Sequential()
    model.add(layers.Dense(30,init = 'uniform', activation = 'relu', input_dim = 49))
    model.add(layers.Dense(10,init = 'uniform', activation = 'relu'))
    model.add(layers.Dense(3, init = 'uniform', activation = 'softmax'))

    model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='ada
    model.fit(nn_train, nn_label, epochs=15, batch_size=500)

    return model
```

In [8]:
```
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 = 3
    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 [9]:
```
# 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)
```

```
Loading  ./sensIT_train.csv  dataset ...

Dataset loaded successfully


Loading  ./sensIT_test.csv  dataset ...

Dataset loaded successfully




/home/shashwati/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel_launcher.py:22: UserW
/home/shashwati/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel_launcher.py:23: UserW
/home/shashwati/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel_launcher.py:24: UserW


Epoch 1/15
20271/20271 [==============================] - 1s 49us/step - loss: 1.0812 - acc: 0.5889
Epoch 2/15
20271/20271 [==============================] - 0s 15us/step - loss: 0.9867 - acc: 0.5978
Epoch 3/15
20271/20271 [==============================] - 0s 16us/step - loss: 0.9305 - acc: 0.5978
Epoch 4/15
20271/20271 [==============================] - 0s 16us/step - loss: 0.9140 - acc: 0.5978
Epoch 5/15
20271/20271 [==============================] - 0s 15us/step - loss: 0.9039 - acc: 0.6023
Epoch 6/15
20271/20271 [==============================] - 0s 18us/step - loss: 0.8977 - acc: 0.6058
Epoch 7/15
20271/20271 [==============================] - 0s 18us/step - loss: 0.8934 - acc: 0.6073
Epoch 8/15
```

```
20271/20271 [==============================] - 0s 15us/step - loss: 0.8909 - acc: 0.6083
Epoch 9/15
20271/20271 [==============================] - 0s 17us/step - loss: 0.8894 - acc: 0.6072
Epoch 10/15
20271/20271 [==============================] - 0s 16us/step - loss: 0.8880 - acc: 0.6073
Epoch 11/15
20271/20271 [==============================] - 0s 19us/step - loss: 0.8865 - acc: 0.6081
Epoch 12/15
20271/20271 [==============================] - 0s 11us/step - loss: 0.8853 - acc: 0.6079
Epoch 13/15
20271/20271 [==============================] - 0s 20us/step - loss: 0.8840 - acc: 0.6073
Epoch 14/15
20271/20271 [==============================] - 0s 16us/step - loss: 0.8829 - acc: 0.6077
Epoch 15/15
20271/20271 [==============================] - 1s 30us/step - loss: 0.8819 - acc: 0.6087
Epoch 1/15
23943/23943 [==============================] - 1s 39us/step - loss: 1.0781 - acc: 0.5265
Epoch 2/15
23943/23943 [==============================] - 0s 10us/step - loss: 1.0132 - acc: 0.5310
Epoch 3/15
23943/23943 [==============================] - 0s 11us/step - loss: 1.0065 - acc: 0.5310
Epoch 4/15
23943/23943 [==============================] - 0s 10us/step - loss: 1.0031 - acc: 0.5316
Epoch 5/15
23943/23943 [==============================] - 0s 10us/step - loss: 0.9934 - acc: 0.5350
Epoch 6/15
23943/23943 [==============================] - 0s 19us/step - loss: 0.9758 - acc: 0.5439
Epoch 7/15
23943/23943 [==============================] - 0s 13us/step - loss: 0.9636 - acc: 0.5490
Epoch 8/15
23943/23943 [==============================] - 0s 10us/step - loss: 0.9578 - acc: 0.5512
Epoch 9/15
23943/23943 [==============================] - 0s 10us/step - loss: 0.9541 - acc: 0.5545
Epoch 10/15
23943/23943 [==============================] - 0s 11us/step - loss: 0.9510 - acc: 0.5562
Epoch 11/15
23943/23943 [==============================] - 0s 13us/step - loss: 0.9483 - acc: 0.5592
Epoch 12/15
23943/23943 [==============================] - 0s 10us/step - loss: 0.9467 - acc: 0.5591
Epoch 13/15
23943/23943 [==============================] - 0s 10us/step - loss: 0.9432 - acc: 0.5601
Epoch 14/15
23943/23943 [==============================] - 0s 14us/step - loss: 0.9413 - acc: 0.5612
Epoch 15/15
23943/23943 [==============================] - 0s 15us/step - loss: 0.9396 - acc: 0.5622
Epoch 1/15
34609/34609 [==============================] - 1s 34us/step - loss: 0.9702 - acc: 0.8717
Epoch 2/15
```

```
34609/34609 [==============================] - 0s 12us/step - loss: 0.5252 - acc: 0.8733
Epoch 3/15
34609/34609 [==============================] - 1s 15us/step - loss: 0.3897 - acc: 0.8733
Epoch 4/15
34609/34609 [==============================] - 1s 19us/step - loss: 0.3500 - acc: 0.8733
Epoch 5/15
34609/34609 [==============================] - 1s 15us/step - loss: 0.3378 - acc: 0.8733
Epoch 6/15
34609/34609 [==============================] - 0s 14us/step - loss: 0.3293 - acc: 0.8733
Epoch 7/15
34609/34609 [==============================] - 1s 15us/step - loss: 0.3227 - acc: 0.8733
Epoch 8/15
34609/34609 [==============================] - 0s 14us/step - loss: 0.3183 - acc: 0.8734
Epoch 9/15
34609/34609 [==============================] - 1s 15us/step - loss: 0.3153 - acc: 0.8754
Epoch 10/15
34609/34609 [==============================] - 1s 14us/step - loss: 0.3130 - acc: 0.8768
Epoch 11/15
34609/34609 [==============================] - 0s 11us/step - loss: 0.3112 - acc: 0.8779
Epoch 12/15
34609/34609 [==============================] - 0s 13us/step - loss: 0.3098 - acc: 0.8799
Epoch 13/15
34609/34609 [==============================] - 0s 12us/step - loss: 0.3085 - acc: 0.8802
Epoch 14/15
34609/34609 [==============================] - 0s 14us/step - loss: 0.3075 - acc: 0.8813
Epoch 15/15
34609/34609 [==============================] - 1s 17us/step - loss: 0.3063 - acc: 0.8819
```

```
In [10]: 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.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
             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] - 1]
                 pred = np.argmax(nn_model_class.predict(x))+1

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

             acc_score = correct/len(test)
```

```
        return confusion_matrix, acc_score
```

```
In [11]: # Predicting
         cm, acc_score = neural_shrub_predict(tree, shrubs, test, test_label)
         print("Confusion Matrix:\n\n", cm)
```

```
Confusion Matrix:

 [[2808 1617  178]
 [ 994 3458  858]
 [ 668 1527 7597]]
```

```
In [12]: def metrics(cm, cls, size):
             cm = np.array(cm)
             tp = cm[cls][cls]
             fp = sum(cm[x, cls] for x in range(3))-cm[cls][cls]
             fn = sum(cm[cls, x] for x in range(3))-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 [13]: # Class 1
         precision0, recall0, f0, acc0 = metrics(cm, 0, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 1: ", round(precision0, 3), "   ", round(recall0, 3), \
               " ", round(f0, 3), "    ", round(acc0,3))
```

```
        Precision Recall F-measure Accuracy
Class 1:  0.628    0.61   0.619     0.825
```

```
In [14]: # Class 2
         precision1, recall1, f1, acc1 = metrics(cm, 1, len(test))
         print("        Precision Recall F-measure Accuracy")
         print("Class 2: ", round(precision1, 3), "   ", round(recall1, 3), \
               " ", round(f1, 3), "    ", round(acc1,3))
```

```
        Precision Recall F-measure Accuracy
Class 2:  0.524    0.651   0.581     0.746
```

```
In [15]: # Class 3
         precision2, recall2, f2, acc2 = metrics(cm, 2, len(test))
```

```
        print("        Precision Recall F-measure Accuracy")
        print("Class 3: ", round(precision2, 3), "  ", round(recall2, 3), \
              " ", round(f2, 3), "    ", round(acc2,3))
```

```
        Precision Recall F-measure Accuracy
Class 3:  0.88    0.776   0.825     0.836
```

```
In [16]: avg_p = (precision0 + precision1 + precision2)/3.0
         avg_r = (recall0 + recall1 + recall2) / 3.0
         avg_f = (f0 + f1 + f2) / 3.0
         avg_a = (acc0 + acc1 + acc2)/ 3.0
         print("        Precision Recall F-measure Accuracy")
         print("Average: ", round(avg_p, 3), "  ", round(avg_r, 3), \
               " ", round(avg_f, 3), "    ", round(avg_a,3))
```

```
        Precision Recall F-measure Accuracy
Average:  0.677    0.679   0.675     0.802
```

```
In [17]: # Number of instances correctly classified
         print("Accuracy_score: ", round(acc_score, 4))
         total_time_taken = dt_end - dt_start + max_time
         print("Training Time: %s secs" % round(total_time_taken, 3))
```

```
Accuracy_score:  0.7035
Training Time: 12.572 secs
```