Homework_1

January 28, 2020

```
In [259]: import numpy as np
          import matplotlib.pyplot as plt
          from matplotlib.colors import ListedColormap
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import confusion_matrix
          from sklearn.metrics import classification_report
          from sklearn.svm import SVC
          from sklearn import preprocessing
          from sklearn import tree
          from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
          from sklearn.naive_bayes import GaussianNB
          from sklearn.neural_network import MLPClassifier
```

1 Load Dataset

```
In [214]: # load dataset
          train_x = np.load("pokemon_train_x.npy")
          train_y = np.load("pokemon_train_y.npy")
```

2 Preprocess training set and testing set

```
In [215]: # Preprocessing Training dataset X
          x_processed = []

          for t in train_x:
              list_x = list(t)
              if list_x[9] == 'F':
                  list_x[9] = 0
              else:
                  list_x[9] = 1
              x_processed.append(list_x)
```

```
In [216]: # Preprocessing Training label set Y
          np.unique(train_y)
          y_processed = []
```

```
        for names in train_y:
            if names == 'Bulbasaur':
                names = 2
            elif names == 'Charmander':
                names = 3
            elif names == 'Gastly':
                names = 7
            elif names == 'Jigglypuff':
                names = 10
            elif names == 'Pidgey':
                names = 9
            elif names == 'Pikachu':
                names = 11
            elif names == 'Squirtle':
                names = 19
            elif names == 'Sudowoodo':
                names = 21
            y_processed.append(names)
```

3 Split the original training set into training set and validation set

```
In [217]: # split training set (80%) and validation set (20%)
          x_train, x_vali, y_train, y_true = train_test_split(x_processed,  \
                                                y_processed, test_size = 0.2)
```

4 KNN Algorithm

```
In [218]: # train KNN model with training dataset with k = 80 = sqrt(8000*0.8)
          neigh = KNeighborsClassifier(n_neighbors=80)
          neigh.fit(x_train, y_train)
          KNeighborsClassifier(...)
          y_pred = neigh.predict(x_vali)
          # confusion matrix
          confusion_matrix(y_true, y_pred)

Out[218]: array([[158,   6,   0,   0,   0,   2,  25,   0],
                 [ 19, 168,   3,  10,   0,  12,   7,   0],
                 [  0,   0, 193,   3,   0,   0,   0,   0],
                 [  2,  11,   1, 191,   0,  15,   1,   0],
                 [  2,   0,   0,   0, 195,   0,   0,   0],
                 [  0,  14,   0,   5,   0, 180,   0,   0],
                 [ 43,   2,   0,   0,   0,   0, 139,   0],
                 [  0,   0,   0,   0,   0,   0,   0, 193]])

In [219]: # classification result
          target_names = list(np.unique(train_y))
          print(classification_report(y_true, y_pred, target_names = target_names))
```

```
             precision    recall  f1-score   support

   Bulbasaur       0.71      0.83      0.76       191
  Charmander       0.84      0.77      0.80       219
      Gastly       0.98      0.98      0.98       196
  Jigglypuff       0.91      0.86      0.89       221
      Pidgey       1.00      0.99      0.99       197
     Pikachu       0.86      0.90      0.88       199
    Squirtle       0.81      0.76      0.78       184
   Sudowoodo       1.00      1.00      1.00       193

   micro avg       0.89      0.89      0.89      1600
   macro avg       0.89      0.89      0.89      1600
weighted avg       0.89      0.89      0.89      1600
```

5 SVM Algorithm

```
In [272]: # Normalize the training set
          min_max_scaler = preprocessing.MinMaxScaler()
          x_norm = min_max_scaler.fit_transform(x_train)
          x_vali_norm = min_max_scaler.fit_transform(x_vali)
```

```
In [266]: clf_svm = SVC(gamma = 'auto')
          clf_svm.fit(x_norm, y_train)
```

```
Out[266]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
            max_iter=-1, probability=False, random_state=None, shrinking=True,
            tol=0.001, verbose=False)
```

```
In [268]: y_pred_svm = clf_svm.predict(x_vali_norm)
          confusion_matrix(y_true, y_pred_svm)
```

```
Out[268]: array([[174,   4,   0,   0,   0,   0,  13,   0],
                 [ 15, 176,   0,   0,   0,   0,  28,   0],
                 [  0,   0, 196,   0,   0,   0,   0,   0],
                 [  0,   0,   0, 220,   0,   1,   0,   0],
                 [  0,   0,   0,   0, 186,   0,  11,   0],
                 [  0,   1,   0,   0,   0, 195,   3,   0],
                 [  1,   0,   0,   0,   0,   0, 183,   0],
                 [  0,   0,   0,   0,   0,   0,   0, 193]])
```

```
In [269]: print(classification_report(y_true, y_pred_svm, target_names = target_names))
```

```
             precision    recall  f1-score   support
```

```
   Bulbasaur       0.92      0.91      0.91       191
  Charmander       0.97      0.80      0.88       219
      Gastly       1.00      1.00      1.00       196
  Jigglypuff       1.00      1.00      1.00       221
      Pidgey       1.00      0.94      0.97       197
     Pikachu       0.99      0.98      0.99       199
    Squirtle       0.77      0.99      0.87       184
   Sudowoodo       1.00      1.00      1.00       193

   micro avg       0.95      0.95      0.95      1600
   macro avg       0.96      0.95      0.95      1600
weighted avg       0.96      0.95      0.95      1600
```

6 Decision Tree Algorithm

```
In [231]: clf_tree = tree.DecisionTreeClassifier()
          clf_tree = clf_tree.fit(x_train, y_train)
```

```
In [232]: y_pred_tree = clf_tree.predict(x_vali)
          confusion_matrix(y_true, y_pred_tree)
```

```
Out[232]: array([[173,  13,   0,   0,   4,   0,   1,   0],
                 [ 16, 189,   0,   0,   0,   0,  14,   0],
                 [  0,   0, 196,   0,   0,   0,   0,   0],
                 [  0,   0,   0, 221,   0,   0,   0,   0],
                 [  2,   0,   0,   0, 194,   1,   0,   0],
                 [  0,   1,   0,   0,   1, 197,   0,   0],
                 [  6,   5,   0,   0,   0,   0, 173,   0],
                 [  0,   0,   0,   0,   0,   0,   0, 193]])
```

```
In [237]: print(classification_report(y_true, y_pred_tree, target_names = target_names))
```

```
              precision    recall  f1-score   support

   Bulbasaur       0.88      0.91      0.89       191
  Charmander       0.91      0.86      0.89       219
      Gastly       1.00      1.00      1.00       196
  Jigglypuff       1.00      1.00      1.00       221
      Pidgey       0.97      0.98      0.98       197
     Pikachu       0.99      0.99      0.99       199
    Squirtle       0.92      0.94      0.93       184
   Sudowoodo       1.00      1.00      1.00       193

   micro avg       0.96      0.96      0.96      1600
   macro avg       0.96      0.96      0.96      1600
weighted avg       0.96      0.96      0.96      1600
```

7 LDA Algorithm

```
In [235]: clf_lda = LinearDiscriminantAnalysis()
          clf_lda = clf_lda.fit(x_train, y_train)
```

```
In [236]: y_pred_lda = clf_lda.predict(x_vali)
          confusion_matrix(y_true, y_pred_lda)
```

```
Out[236]: array([[166,  24,   0,   0,   0,   0,   1,   0],
                 [ 12, 200,   0,   0,   0,   0,   7,   0],
                 [  0,   0, 196,   0,   0,   0,   0,   0],
                 [  0,   0,   0, 221,   0,   0,   0,   0],
                 [  3,  17,   0,   0, 173,   0,   4,   0],
                 [  0,   5,   0,   0,   0, 190,   4,   0],
                 [  4,  11,   0,   0,   0,   0, 169,   0],
                 [  0,   0,   0,   0,   0,   0,   0, 193]])
```

```
In [238]: print(classification_report(y_true, y_pred_lda, target_names = target_names))
```

```
              precision    recall  f1-score   support

   Bulbasaur       0.90      0.87      0.88       191
  Charmander       0.78      0.91      0.84       219
      Gastly       1.00      1.00      1.00       196
  Jigglypuff       1.00      1.00      1.00       221
      Pidgey       1.00      0.88      0.94       197
     Pikachu       1.00      0.95      0.98       199
    Squirtle       0.91      0.92      0.92       184
   Sudowoodo       1.00      1.00      1.00       193

   micro avg       0.94      0.94      0.94      1600
   macro avg       0.95      0.94      0.94      1600
weighted avg       0.95      0.94      0.94      1600
```

8 Naive Bayes Algorithm

```
In [241]: clf_nb = GaussianNB()
          clf_nb = clf_nb.fit(x_train, y_train)
```

```
In [242]: y_pred_nb = clf_nb.predict(x_vali)
          confusion_matrix(y_true, y_pred_nb)
```

```
Out[242]: array([[159,  15,   0,   0,  17,   0,   0,   0],
                  [ 32, 146,   0,   0,   4,   0,  37,   0],
                  [  0,   0, 196,   0,   0,   0,   0,   0],
                  [  0,   0,   0, 221,   0,   0,   0,   0],
                  [  1,   0,   0,   0, 195,   1,   0,   0],
                  [  0,   0,   0,   0,  29, 170,   0,   0],
                  [  1,  23,   0,   0,   0,   0, 160,   0],
                  [  0,   0,   0,   0,   0,   0,   0, 193]])
```

```
In [243]: print(classification_report(y_true, y_pred_nb, target_names = target_names))
```

```
              precision    recall  f1-score   support

   Bulbasaur       0.82      0.83      0.83       191
  Charmander       0.79      0.67      0.72       219
      Gastly       1.00      1.00      1.00       196
  Jigglypuff       1.00      1.00      1.00       221
      Pidgey       0.80      0.99      0.88       197
     Pikachu       0.99      0.85      0.92       199
    Squirtle       0.81      0.87      0.84       184
   Sudowoodo       1.00      1.00      1.00       193

   micro avg       0.90      0.90      0.90      1600
   macro avg       0.90      0.90      0.90      1600
weighted avg       0.90      0.90      0.90      1600
```

9 Neural Network Algorithm

```
In [252]: clf_mlp = MLPClassifier(solver = 'adam', alpha = 1e-5,  \
                                  hidden_layer_sizes = (32,16,8), \
                                     activation = 'relu')
          clf_mlp = clf_mlp.fit(x_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/sklearn/neural_ne
  % self.max_iter, ConvergenceWarning)
```

```
In [254]: y_pred_mlp = clf_mlp.predict(x_vali)
          confusion_matrix(y_true, y_pred_mlp)
```

```
Out[254]: array([[186,   3,   0,   0,   0,   0,   2,   0],
                  [  3, 207,   0,   0,   0,   7,   2,   0],
                  [  0,   0, 196,   0,   0,   0,   0,   0],
                  [  0,   1,   0, 219,   0,   1,   0,   0],
                  [  0,   0,   0,   0, 197,   0,   0,   0],
                  [  0,   8,   0,   0,   0, 191,   0,   0],
                  [ 17,   4,   0,   0,   0,   0, 163,   0],
                  [  0,   0,   0,   0,   0,   0,   0, 193]])
```

```
In [255]: print(classification_report(y_true, y_pred_mlp, target_names = target_names))
```

```
              precision    recall  f1-score   support

   Bulbasaur       0.90      0.97      0.94       191
  Charmander       0.93      0.95      0.94       219
      Gastly       1.00      1.00      1.00       196
  Jigglypuff       1.00      0.99      1.00       221
      Pidgey       1.00      1.00      1.00       197
     Pikachu       0.96      0.96      0.96       199
    Squirtle       0.98      0.89      0.93       184
   Sudowoodo       1.00      1.00      1.00       193

   micro avg       0.97      0.97      0.97      1600
   macro avg       0.97      0.97      0.97      1600
weighted avg       0.97      0.97      0.97      1600
```

10 Testing

```
In [273]: # load testing dataset
          test = np.load("pokemon_test_x.npy")
          # preprocess testing dataset
          test_processed = []
          for t in test:
              list_t = list(t)
              if list_t[9] == 'F':
                  list_t[9] = 0
              else:
                  list_t[9] = 1
              test_processed.append(list_t)
          test_pred = clf_tree.predict(test_processed)
```

```
In [ ]:
```