ml-ex4

August 4, 2017

0.1 Neural networks learning

This exercise is described in ex4.pdf.

```
In [1]: import scipy.io as sio
        import pandas as pd
        import matplotlib.pyplot as plt

        from sklearn.neural_network import MLPClassifier
        from sklearn.model_selection import StratifiedShuffleSplit, GridSearchCV

        %matplotlib inline
```

```
In [2]: # Load the hand-written digits dataset
        digits = sio.loadmat('data/ml-ex4/ex4data1.mat')
```

```
In [3]: # Digit image data (5000 images with 400 features/pixels)
        X = digits['X']

        # Digit classes (1-10) where digit 0 is assigned class 10
        y = digits['y'].ravel()
```

Sample images are shown in Exercise 3 notebook.

```
In [4]: # Create a feed-forward neural network with a single hidden layer
        # with 25 logistic units. Input layer size is derived from number
        # of features in X. Output layer size is derived from number of
        # classes in y.
        clf = MLPClassifier((25,), activation='logistic', solver='lbfgs')
        clf.fit(X, y)
```

```
Out[4]: MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
               beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
               hidden_layer_sizes=(25,), learning_rate='constant',
               learning_rate_init=0.001, max_iter=200, momentum=0.9,
               nesterovs_momentum=True, power_t=0.5, random_state=None,
               shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
               verbose=False, warm_start=False)
```

In [5]:
```
# Classification accuracy on training set
clf.score(X, y)
```

Out[5]:
```
1.0
```

In [12]:
```
# For cross validation, use two stratified randomized folds
# where the test set size = 0.1 * dataset size. Statified
# means that the folds are made by preserving the percentage
# of samples for each class.
cv = StratifiedShuffleSplit(2, test_size=0.1, random_state=0)

# Run a grid search to find the best value values for regularization
# parameter alpha using the cross validator (cv) defined above.
gs = GridSearchCV(clf, param_grid={'alpha':[1e-2, 1e-1, 1e0, 1e1]}, cv=cv)
gs.fit(X, y)
```

Out[12]:
```
GridSearchCV(cv=StratifiedShuffleSplit(n_splits=2, random_state=0, test_size=0.1,
            train_size=None),
       error_score='raise',
       estimator=MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(25,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'alpha': [0.01, 0.1, 1.0, 10.0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

In [7]:
```
# Show the cross validation results in a pandas data frame
# - column mean_train_score: mean classification accuracy on training set
# - column mean_test_score: mean classification accuracy on test set
pd.DataFrame(gs.cv_results_)
```

Out[7]:
```
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0       1.437032         0.001812            0.919          1.000000
1       2.473062         0.001954            0.937          1.000000
2       4.012578         0.002556            0.946          0.997556
3       4.014157         0.002660            0.920          0.943222

  param_alpha          params  rank_test_score  split0_test_score  \
0        0.01  {'alpha': 0.01}                4              0.910
1         0.1   {'alpha': 0.1}                2              0.924
2           1   {'alpha': 1.0}                1              0.936
3          10  {'alpha': 10.0}                3              0.912

   split0_train_score  split1_test_score  split1_train_score  std_fit_time  \
```

```
0         1.000000           0.928          1.000000      0.004382
1         1.000000           0.950          1.000000      0.046205
2         0.997333           0.956          0.997778      0.067075
3         0.942889           0.928          0.943556      0.046254

   std_score_time  std_test_score  std_train_score
0        0.000024           0.009         0.000000
1        0.000198           0.013         0.000000
2        0.000335           0.010         0.000222
3        0.000469           0.008         0.000333
```

In [8]:
```
# Best classifier is the neural network with alpha=1.0
clf_best = gs.best_estimator_
clf_best
```

Out[8]:
```
MLPClassifier(activation='logistic', alpha=1.0, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(25,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

In [9]:
```
# Classification accuracy on training set
clf_best.score(X, y)
```

Out[9]:
```
0.99739999999999995
```

In [10]:
```
# Obtain weight matrix between input and hidden layer
# (input layer size = 400, hidden layer size = 25)
Theta1 = clf_best.coefs_[0].T
Theta1.shape
```

Out[10]:
```
(25, 400)
```

In [11]:
```
# Visualize input weights into hidden units (400 per hidden unit)
n_rows = 5
n_cols = 5

plt.subplots_adjust(top=.9, hspace=.4)
plt.figure(figsize=(1.8 * n_cols, 2.4 * n_rows))

for i, row in enumerate(Theta1):
    plt.subplot(n_rows, n_cols, i + 1)
    plt.imshow(row.reshape((20,20), order='F'), cmap=plt.cm.gray)
    plt.title(f'hidden unit {i + 1}')
    plt.xticks(())
    plt.yticks(())
```

```
<matplotlib.figure.Figure at 0x10684c860>
```