ml-ex3

August 4, 2017

0.1 Multi-class classification

This exercise is described in ex3.pdf.

```
In [3]: import numpy as np
        import scipy as sp
        import scipy.io as sio
        import matplotlib.pyplot as plt

        from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
        from sklearn.metrics import classification_report

        %matplotlib inline
```

0.1.1 Hand-written digits dataset

```
In [4]: # Load the hand-written digits dataset
        digits = sio.loadmat('data/ml-ex3/ex3data1.mat')
```

```
In [5]: # 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()
```

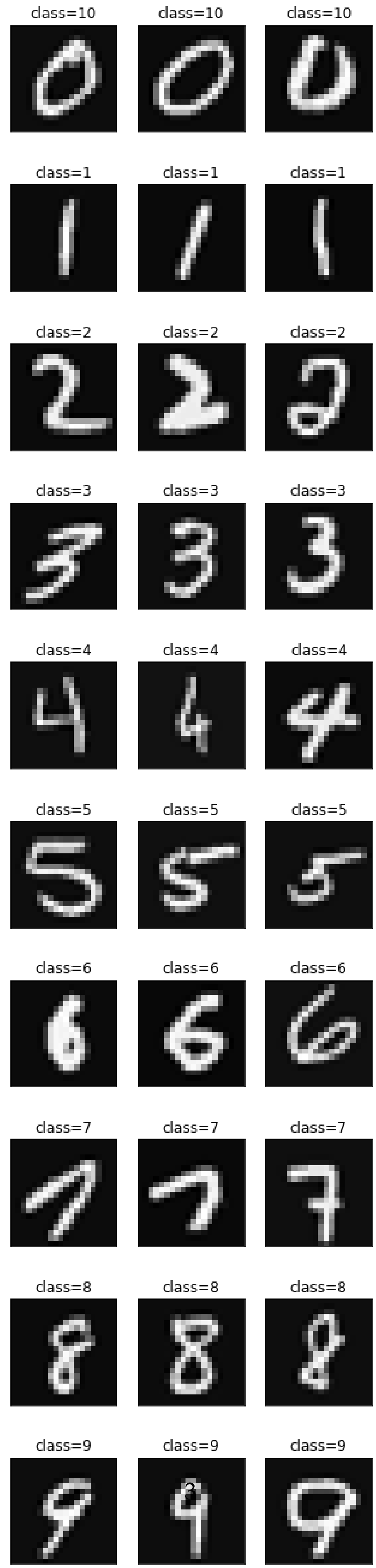
```
In [6]: # Plot three samples from each class
        n_rows = 10
        n_cols = 3

        selected = np.linspace(0, X.shape[0] - 1, n_rows * n_cols, dtype='int16')

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

        for i, idx in enumerate(selected):
            plt.subplot(n_rows, n_cols, i + 1)
            plt.imshow(X[idx].reshape((20,20), order='F'), cmap=plt.cm.gray)
            plt.title(f'class={y[idx]}')
            plt.xticks(())
            plt.yticks(())
```

<matplotlib.figure.Figure at 0x107172ba8>


class=10
class=10
class=10
class=1
class=1
class=1
class=2
class=2
class=2
class=3
class=3
class=3
class=4
class=4
class=4
class=5
class=5
class=5
class=6
class=6
class=6
class=7
class=7
class=7
class=8
class=8
class=8
class=9
class=9
class=9

0.1.2 Logistic regression

```
In [7]: # Train a logistic regression classifier with C=1.0 (inverse regularization strength)
        clf = LogisticRegression(C=1.0, solver='newton-cg')
        clf.fit(X, y)
```

```
Out[7]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                  penalty='l2', random_state=None, solver='newton-cg', tol=0.0001,
                  verbose=0, warm_start=False)
```

```
In [8]: # Classification accuracy on training set
        clf.score(X, y)
```

```
Out[8]: 0.9446
```

```
In [9]: # Train a logistic regression classifier, running
        # built-in cross validation for selecting the best
        # C value
        clf_cv = LogisticRegressionCV(Cs=[1e-1, 1e0, 1e1], solver='newton-cg')
        clf_cv.fit(X, y)
```

```
Out[9]: LogisticRegressionCV(Cs=[0.1, 1.0, 10.0], class_weight=None, cv=None,
                   dual=False, fit_intercept=True, intercept_scaling=1.0,
                   max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2',
                   random_state=None, refit=True, scoring=None, solver='newton-cg',
                   tol=0.0001, verbose=0)
```

```
In [10]: # Classification accuracy on training set
         # using the model that was selected during
         # cross-validation
         clf_cv.score(X, y)
```

```
Out[10]: 0.94079999999999997
```

0.1.3 Neural networks

```
In [11]: # Load the parameters of a pre-trained neural network
         weights = sio.loadmat('data/ml-ex3/ex3weights.mat')

         Theta1 = weights['Theta1']
         Theta2 = weights['Theta2']

         Theta1.shape, Theta2.shape
```

```
Out[11]: ((25, 401), (10, 26))
```

In [12]:
```
# Number of samples (first dimension of design matrix X)
n_samples = X.shape[0]

# sigmoid function
sigmoid = sp.special.expit
```

In [13]:
```
# Input layer data
A1 = np.c_[np.ones(n_samples), X]
```

In [14]:
```
# Hidden layer activations
Z2 = A1.dot(Theta1.T)
A2 = np.c_[np.ones(n_samples), sigmoid(Z2)]
```

In [15]:
```
# Output layer activations
Z3 = A2.dot(Theta2.T)
A3 = sigmoid(Z3)
```

In [16]:
```
# Predicted class is index of highest value per row + 1
y_pred = np.argmax(A3, axis=1) + 1
```

In [17]:
```
# Print classification report showing the main classification metrics
print(classification_report(y, y_pred, target_names=['1','2','3','4','5','6','7','8','9
```

```
             precision    recall  f1-score   support

          1     0.9684    0.9820    0.9752       500
          2     0.9818    0.9700    0.9759       500
          3     0.9776    0.9600    0.9687       500
          4     0.9699    0.9680    0.9690       500
          5     0.9723    0.9840    0.9781       500
          6     0.9782    0.9860    0.9821       500
          7     0.9778    0.9700    0.9739       500
          8     0.9781    0.9820    0.9800       500
          9     0.9657    0.9580    0.9618       500
         10     0.9822    0.9920    0.9871       500

avg / total     0.9752    0.9752    0.9752      5000
```