MNISTLogReg

March 28, 2018

```
In [13]: import numpy as np
         import pandas as pd
         from keras.datasets import mnist
         from sklearn.linear_model import LogisticRegression
         from sklearn import metrics

         import matplotlib.pyplot as plt
         import seaborn as sns

         %matplotlib inline
```

1 Logistic regression for MNIST

Logistic regression is a model that has many interpretations. The one I will use is the idea that logistic regression minimizes the cross-entropy between predictions of a linear function passed through a normalizing function and the true probabilities. The cross entropy loss for a single prediction over C categories is as follows:

$$L(y, \hat{y}) = \sum_{k=1}^{C} y_k \log(\hat{y}_k) \tag{1}$$

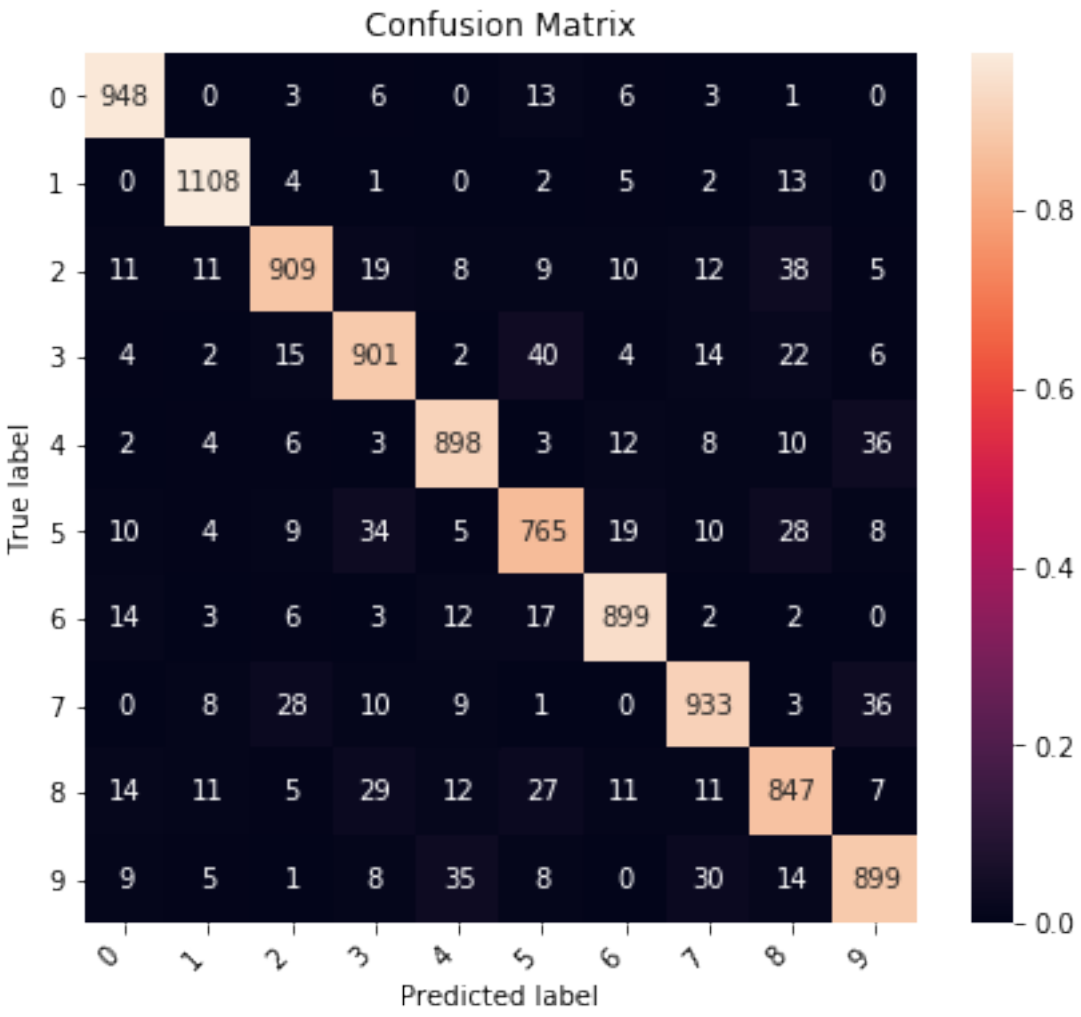
If we have more examples, we just take the average over them. A linear regression model will not suffice here, because $\hat{y}$ are predicted probabilities, so they must be normalized. This can be done by using a softmax function. If we say we get some $u = Wx$, where x is an input vector and W maps x to $\mathbb{R}^C$, then

$$\hat{y}_j = \text{softmax}(u)_j = \frac{e^{u_j}}{\sum_{k=1}^{C} e^{u_k}} \tag{2}$$

Due to memory constraints, we subsample a set of 10000 examples for training this model. We learn the weight matrix W that minimizes the average cross-entropy loss. We can see that we get an accuracy of about 91 percent. A confusion matrix demonstrating the complete test results can be seen at the bottom.

```
In [2]: (x_train, y_train), (x_test, y_test) = mnist.load_data()
        x_train = x_train.reshape(-1, 28**2) / 255.0
        x_test = x_test.reshape(-1, 28**2) / 255.0
```

```
In [3]: # subsample x_train
        n_train_per_label = 1000
        new_examples = []
        for i in range(10):
            subset = x_train[y_train==i]
            choices = np.random.choice(len(subset), n_train_per_label, False)
            new_examples.append(subset[choices])
        x_train = np.concatenate(new_examples, 0)
        y_train = np.repeat(np.arange(10), n_train_per_label)

In [4]: model = LogisticRegression(multi_class='multinomial', solver='lbfgs', verbose=True, ma

In [5]: model.fit(x_train, y_train)
        pred_y = model.predict(x_test)

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    6.9s finished


Out[5]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, max_iter=500, multi_class='multinomial',
                  n_jobs=1, penalty='l2', random_state=None, solver='lbfgs',
                  tol=0.0001, verbose=True, warm_start=False)

In [6]: print('accuracy=%.2f'%model.score(x_test, y_test))

accuracy=0.91


In [24]: conf_mat = metrics.confusion_matrix(y_test, pred_y)
         class_names = list(range(10))
         cm = conf_mat.astype('float') / conf_mat.sum(axis=1)[:, np.newaxis]
         df_cm = pd.DataFrame(
             cm, index=class_names, columns=class_names,
         )
         fig = plt.figure(figsize=(7, 6))
         heatmap = sns.heatmap(df_cm, annot=conf_mat, fmt="d")
         heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
         heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
         plt.ylabel('True label')
         plt.xlabel('Predicted label')
         plt.title('Confusion Matrix')

Out[24]: Text(0.5,1,'Confusion Matrix')
```


Confusion Matrix
True label
Predicted label
0 948 0 3 6 0 13 6 3 1 0
1 0 1108 4 1 0 2 5 2 13 0
2 11 11 909 19 8 9 10 12 38 5
3 4 2 15 901 2 40 4 14 22 6
4 2 4 6 3 898 3 12 8 10 36
5 10 4 9 34 5 765 19 10 28 8
6 14 3 6 3 12 17 899 2 2 0
7 0 8 28 10 9 1 0 933 3 36
8 14 11 5 29 12 27 11 11 847 7
9 9 5 1 8 35 8 0 30 14 899
0 1 2 3 4 5 6 7 8 9
0.8
0.6
0.4
0.2
0.0