LinearClassification-Copy1

November 29, 2015

0.1 Exercise 1: Calculate probability of class 1

Compute the probability of class 1 given the data and the parameters.
arguments: * X: data * W: weight matrix, part of the parameters * b: bias, part of the parameters
returns: * *rate*: probabiliy of the predicted class 1

```
In [37]: def pred(X, W, b):
             return 1 / (1 + np.exp(b + np.sum(W*X, axis=1)))
```

0.2 Exercise 2: Calculate the log-likelihood given the target

Compute the logarithm of the likelihood for logistic regression. The negative log-likelihood is our loss function.
arguments: * X: data * Z: target * W: weight matrix, part of the parameters * b: bias, part of the parameters
returns: * *log likelihood*: logarithm of the likelihood

```
In [38]: def loglikelihood(X, Z, W, b):
             sig = pred(X, W, b)
             return np.sum([Z, 1-Z] * np.log([sig, 1-sig]))
```

0.3 Exercise 3: Implement the gradient of the loss/log-likelihood

Compute the gradient of the loss with respect to the parameters
arguments: * X: data * Z: target * W: weight matrix, part of the parameters * b: bias, part of the parameters
returns: * $dLdW$: gradient of loss wrt to W * $dLdb$: gradient of loss wrt to b

```
In [56]: # gradient of negative log-likelihood

         def grad(X, Z, W, b):
             d = Z - pred(X, W, b)
             return -np.sum(d.ravel()[:,np.newaxis]*X, axis=0), np.sum(d)
```

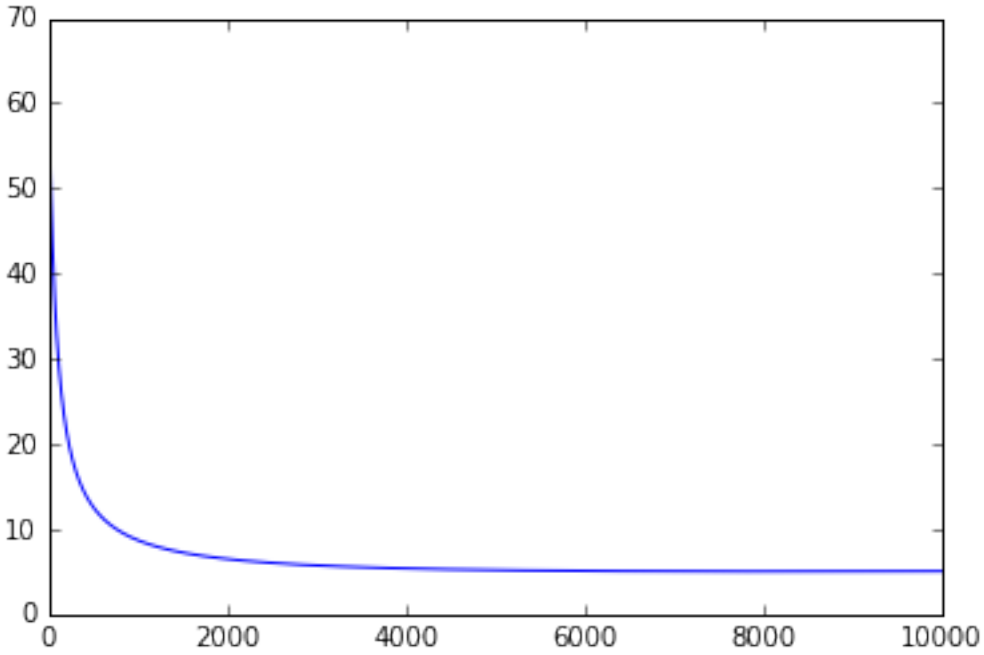
0.4 Exercise 4: Test everything

Run the provied simple gradient descent algorithm to optimize the model parameters and plot the resuling decision boundary.

```
In [60]: W = np.random.randn(1,2) * 0.01
         b = np.random.randn(1) * 0.01

         learning_rate = 0.001
         train_loss = []
         validation_loss = []
```

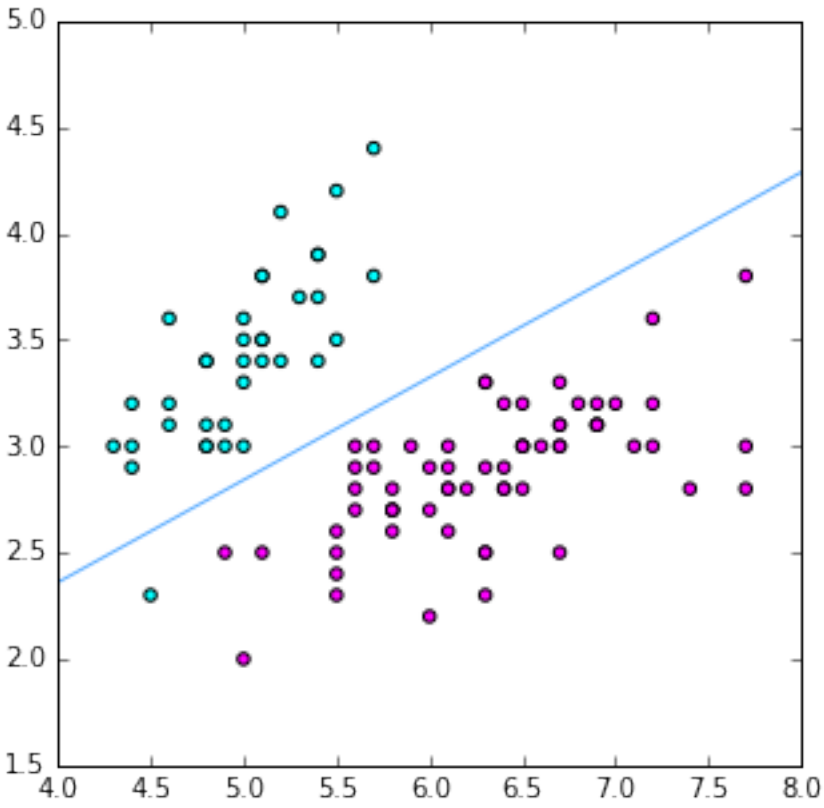
```
        for i in range(10000):
            dLdW, dLdb = grad(X, Z, W, b)

            W += learning_rate * dLdW
            b += learning_rate * dLdb
            train_loss.append( - loglikelihood(X, Z, W, b).mean())

        print('W: %s' % repr(W))
        print('b: %f' % b)
        _ = plt.plot(train_loss)
```

```
W: array([[ -4.98796086,  10.34532045]])
b: -4.453257
```



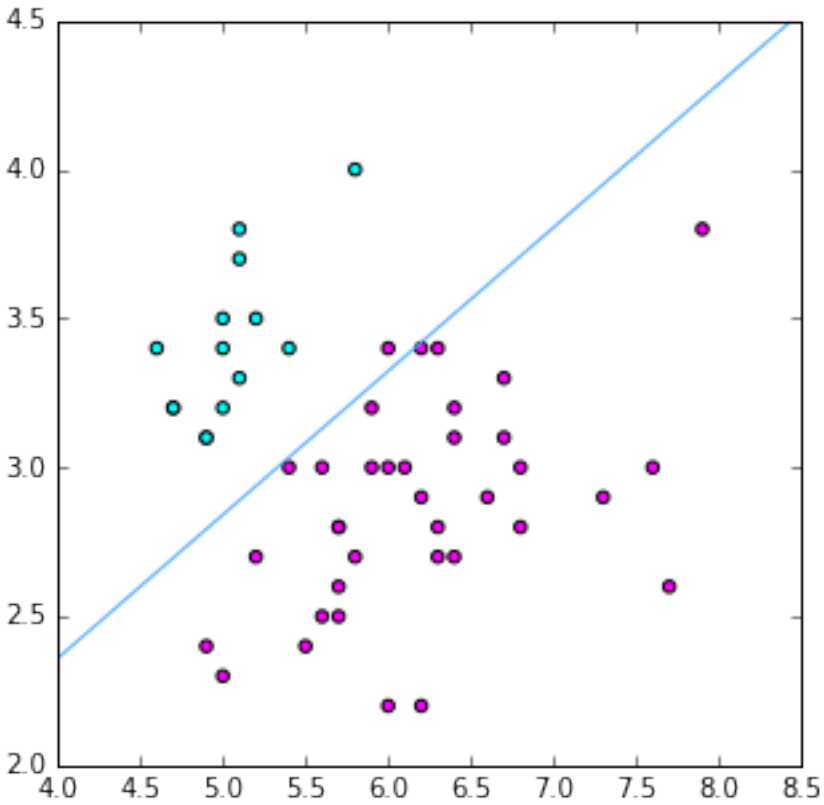
0.4.1 Decision boundary on the training set

```
In [61]: plot_decision_boundary(X, Z, W=W, b=b)
```



0.4.2 Decision boundary on the test set

In [62]: `plot_decision_boundary(XT, ZT, W=W, b=b)`



In [ ]: