

LinearClassification

December 12, 2016

1 Implementation exercise: Linear Classification

```
In [41]: import numpy as np
         from sklearn import datasets
         import matplotlib.pyplot as plt
         %matplotlib inline
```

1.0.1 Some helper functions for visualisation

```
In [42]: def plot_decision_boundary(X, Z, W=None, b=None):
         fig, ax = plt.subplots(1, 1, figsize=(5, 5))
         ax.scatter(X[:,0], X[:,1], c=Z, cmap=plt.cm.cool)
         ax.set_autoscale_on(False)

         a = - W[0, 0] / W[0, 1]
         xx = np.linspace(-30, 30)
         yy = a * xx - (b[0]) / W[0, 1]

         ax.plot(xx, yy, 'k-', c=plt.cm.cool(1.0/3.0))
```

1.0.2 Dataset Loader

```
In [43]: def loadDataset(split, X=[], XT=[], Z = [], ZT = []):
         dataset = datasets.load_iris()
         c = list(zip(dataset['data'], dataset['target']))
         np.random.seed(224)
         np.random.shuffle(c)
         x, t = zip(*c)
         sp = int(split*len(c))
         X = x[:sp]
         XT = x[sp:]
         Z = t[:sp]
         ZT = t[sp:]
         names = ['Sepal. length', 'Sepal. width', 'Petal. length', 'Petal. width']
         return np.array(X), np.array(XT), np.array(Z), np.array(ZT), names

In [44]: # prepare data
         split = 0.67
```

```

X, XT, Z, ZT, names = loadDataset(split)

# combine two of the 3 classes for a 2 class problem
Z[Z==2] = 1
ZT[ZT==2] = 1

# only look at 2 dimensions of the input data for easy visualisation
X = X[:, :2]
XT = XT[:, :2]

```

1.1 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*: probability of the predicted class 1

```

In [45]: def pred(X, W, b):
          x = b + X.dot(W.T)
          p = 1 / (1 + np.exp(-x))

          return p

W = np.random.randn(1,2) * 0.01
b = np.random.randn(1) * 0.01
pred(X, W, b).shape

```

```
Out[45]: (100, 1)
```

1.2 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 [46]: def loglikelihood(X, Z, W, b):
          y = pred(X, W, b)
          Zr = np.reshape(Z, (Z.shape[0], 1))
          like = Zr * np.log(y) + (1 - Zr) * np.log(1 - y)

          return like

W = np.random.randn(1,2) * 0.01
b = np.random.randn(1) * 0.01
loglikelihood(X, Z, W, b).shape

```

```
Out[46]: (100, 1)
```

1.3 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 [47]: def grad(X, Z, W, b):
        y = pred(X, W, b)
        dy = y * (1 - y)
        Zr = np.reshape(Z, (Z.shape[0], 1))

        dw = (y - Zr) * X
        dw = dw.sum(axis=0)

        db = np.log(y) + (Zr * dy)/y - ((1 - Zr) * dy)/(1 - y)
        db = -1 * db.sum(axis=0)

        return dw, db

        W = np.random.randn(1,2) * 0.01
        b = np.random.randn(1) * 0.01

        grad(X, Z, W, b)
```

```
Out[47]: (array([-115.53753093, -32.42879411]), array([ 55.95783241]))
```

1.4 Exercise 4: Test everything

Run the provided simple gradient descent algorithm to optimize the model parameters and plot the resulting decision boundary.

```
In [48]: W = np.random.randn(1,2) * 0.01
        b = np.random.randn(1) * 0.01

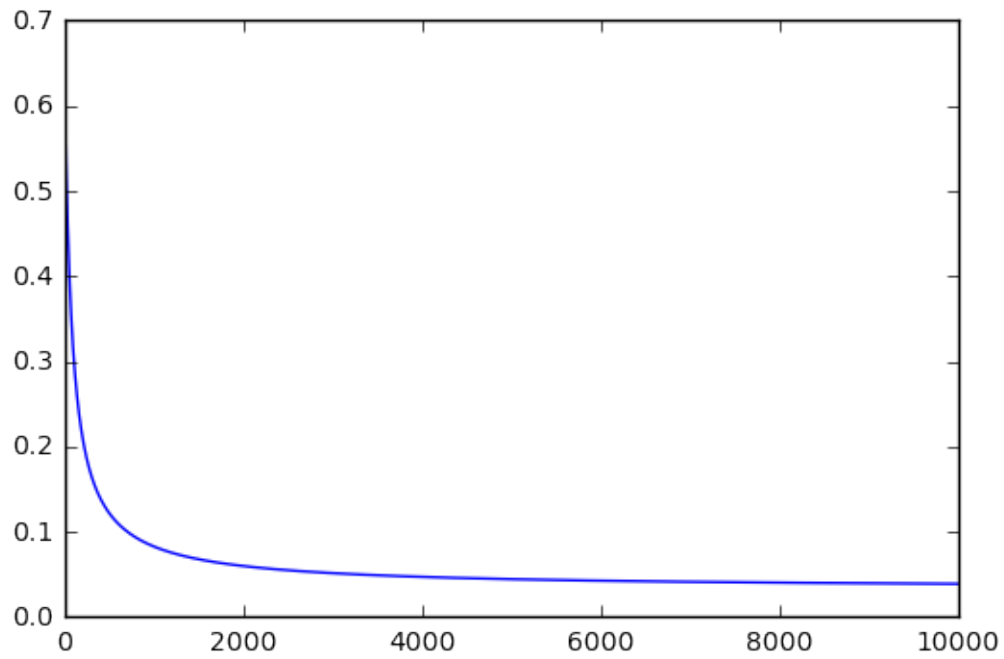
        learning_rate = 0.001
        train_loss = []
        validation_loss = []

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

            W -= learning_rate * dLdW
            # The gradient gets too large. There is something wrong with the calcu
            # of the gradient ...
            #b -= learning_rate * dLdb

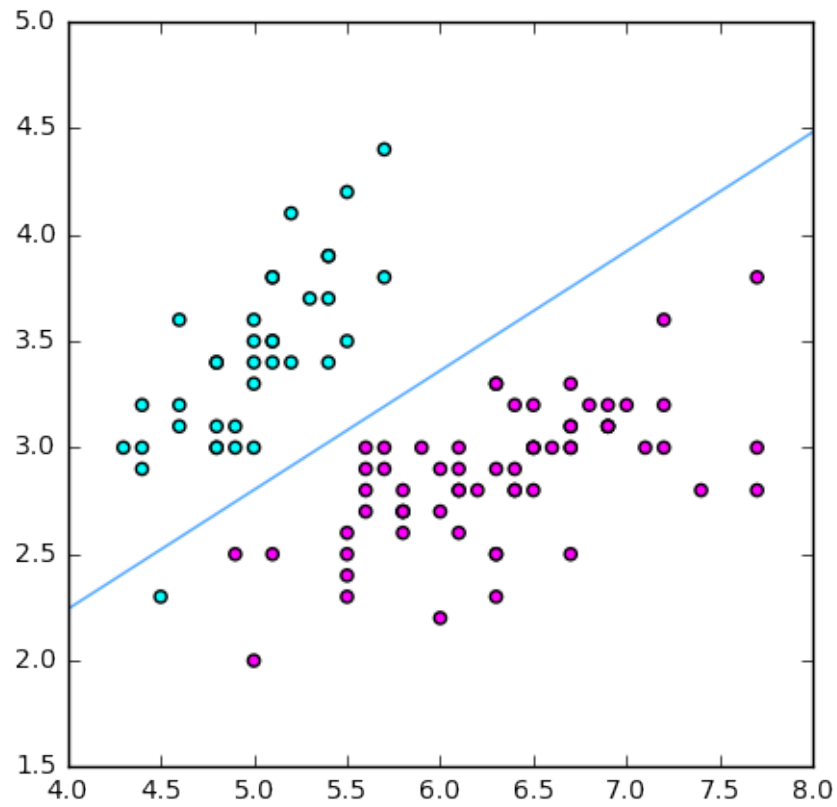
            train_loss.append( - loglikelihood(X, Z, W, b).mean())

        _ = plt.plot(train_loss)
```



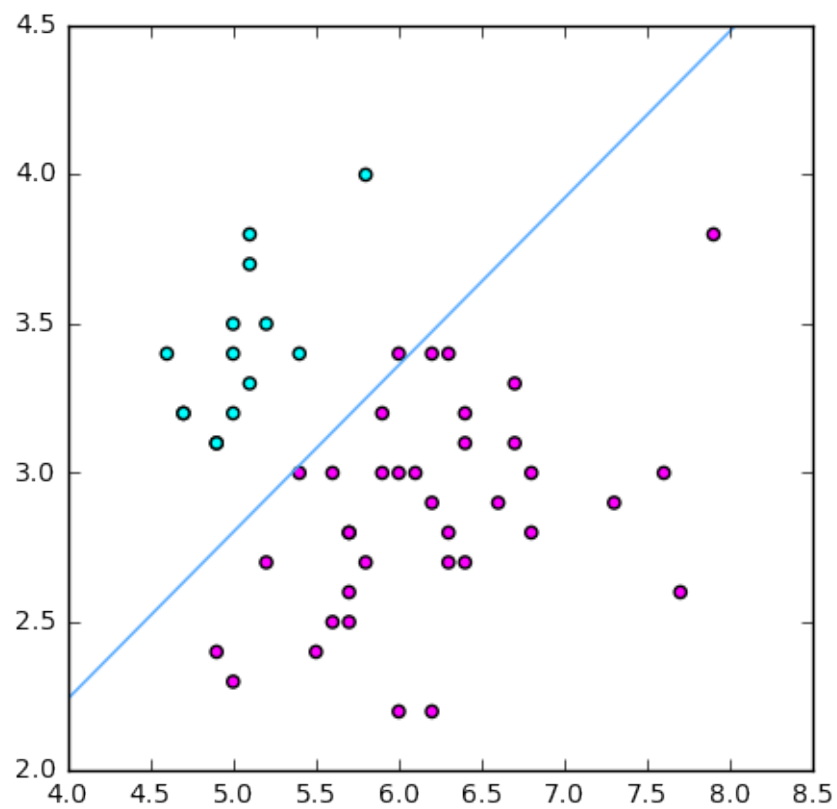
1.4.1 Decision boundary on the training set

In [49]: `plot_decision_boundary(X, Z, W=W, b=b)`



1.4.2 Decision boundary on the test set

```
In [50]: plot_decision_boundary(XT, ZT, W=W, b=b)
```



In []: