Problem Set #3

Joe Michail
DATA SCI 423 – Machine Learning
Northwestern University

May 2, 2020

Question 6.5

The Cross Entropy cost is defined as:

$$g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p \log(\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})) + (1-y_p)\log(1-\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})).$$

Making the substitution that $\sigma(-x) = 1-\sigma(x)$:

$$g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p \log(\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})) + (1-y_p)\log(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})).$$

Splitting the second term and combining like terms:

$$g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p \log(\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})) - y_p\log(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})) + \log(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})).$$

Using the property of logs:

$$g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p \log\left(\frac{\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})}{\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})}\right) + \log(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})).$$

The log-quotient of sigmoid functions in the first term reduces to the argument:

$$g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P} y_p\mathring{\mathbf{x}}_p^T\mathbf{w} + \log(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})).$$

Taking the gradient with respect to $\mathbf{w}$:

$$\begin{aligned}\nabla_{\mathbf{w}} g(\mathbf{w}) &= -\frac{1}{P}\sum_{p=1}^{P} \nabla(y_p\mathring{\mathbf{x}}_p^T\mathbf{w}) + \nabla(\log(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})) \\ &= -\frac{1}{P}\sum_{p=1}^{P} y_p\mathring{\mathbf{x}}_p + \frac{\nabla(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w}))}{\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})}.\end{aligned}$$

Using the property of sigmoids:

$$\frac{d\sigma(y(x))}{dx} = \sigma(y(x))(1-\sigma(y(x))\frac{dy}{dx} = \sigma(y(x))\sigma(-y(x))\frac{dy}{dx}.$$

Therefore:

$$\begin{aligned}\nabla g(\mathbf{w}) &= -\frac{1}{P}\sum_{p=1}^{P} y_p \mathring{\mathbf{x}}_p + \frac{\nabla(\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w}))}{\sigma(-\mathring{\mathbf{x}}_p^T\mathbf{w})} \\ &= -\frac{1}{P}\sum_{p=1}^{P} y_p \mathring{\mathbf{x}}_p - \sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})\mathring{\mathbf{x}}_p.\end{aligned}$$

$$\boxed{\nabla g(\mathbf{w}) = -\frac{1}{P}\sum_{p=1}^{P}\left(y_p - \sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})\right)\mathring{\mathbf{x}}_p}$$

To find the Hessian, we take the gradient again:

$$\begin{aligned}\nabla^2 g(\mathbf{w}) = \nabla_{\mathbf{w}}^T \nabla g(\mathbf{w}) &= -\frac{1}{P}\sum_{p=1}^{P} \nabla_{\mathbf{w}}^T(y_p\mathring{\mathbf{x}}_p) - \nabla_{\mathbf{w}}^T(\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})\mathring{\mathbf{x}}_p) \\ &= \frac{1}{P}\sum_{p=1}^{P} \nabla_{\mathbf{w}}^T(\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w}))\mathring{\mathbf{x}}_p.\end{aligned}$$

Again using the property of sigmoids above:

$$\boxed{\nabla^2 g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w})(1-\sigma(\mathring{\mathbf{x}}_p^T\mathbf{w}))\mathring{\mathbf{x}}_p\mathring{\mathbf{x}}_p^T}$$

Question 6.10

The perceptron cost is defined as:

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\max(0, -y_p\mathring{\mathbf{x}}_p^T\mathbf{w}).$$

The geometric definition of concavity is:

$$g(\lambda\mathbf{w_1} + (1-\lambda)\mathbf{w_2}) \leq \lambda g(\mathbf{w_1}) + (1-\lambda)g(\mathbf{w_2}), \lambda \in [0,1].$$

Combining them we get:

$$\frac{1}{P}\sum_{p=1}^{P}\max(0, -y_p\mathring{\mathbf{x}}_p^T(\lambda\mathbf{w_1} + (1-\lambda)\mathbf{w_2})) \leq \frac{\lambda}{P}\sum_{p=1}^{P}\max(0, -y_p\mathring{\mathbf{x}}_p^T\mathbf{w_1}) + \frac{1-\lambda}{P}\sum_{p=1}^{P}\max(0, -y_p\mathring{\mathbf{x}}_p^T\mathbf{w_2}).$$

Working with the left hand side, the second argument of max can be written as:

$$-y_p\mathring{\mathbf{x}}_p^T(\lambda\mathbf{w_1}+(1-\lambda)\mathbf{w_2}) = -y_p\mathring{\mathbf{x}}_p^T(\lambda(\mathbf{w_1}-\mathbf{w_2})+\mathbf{w_2})$$

If $\mathbf{w_1}$ and $\mathbf{w_2}$ lie just before and after the point of concavity, respectively, then $||\mathbf{w_1}-\mathbf{w_2}||_2 \sim \epsilon$. We can also write $\mathbf{w_1}-\mathbf{w_2}=\vec{\epsilon}$; since λ is bounded between 0 and 1:

$$-y_p\mathring{\mathbf{x}}_p^T(\lambda(\mathbf{w_1}-\mathbf{w_2})+\mathbf{w_2}) = -y_p\mathring{\mathbf{x}}_p^T(\lambda\vec{\epsilon}+\mathbf{w_2}) \approx -y_p\mathring{\mathbf{x}}_p^T\mathbf{w_2}.$$

Going back to the original equation:

$$\sum_{p=1}^{P}\max(0,-y_p\mathring{\mathbf{x}}_p^T\mathbf{w_2}) \leq \lambda\sum_{p=1}^{P}\max(0,-y_p\mathring{\mathbf{x}}_p^T\mathbf{w_1}) + (1-\lambda)\sum_{p=1}^{P}\max(0,-y_p\mathring{\mathbf{x}}_p^T\mathbf{w_2}).$$

Shifting terms:

$$\lambda\sum_{p=1}^{P}\max(0,-y_p\mathring{\mathbf{x}}_p^T\mathbf{w_2}) \leq \lambda\sum_{p=1}^{P}\max(0,-y_p\mathring{\mathbf{x}}_p^T\mathbf{w_1})$$

Since the initial constraint that was put on the problem was $\mathbf{w}_1-\mathbf{w}_2=\vec{\epsilon}$ where $||\epsilon|| << 1$, so we can write $\mathbf{w}_1 \approx \mathbf{w}_2$. Therefore, the terms within the sum are equivalent leaving:

$$\lambda \leq \lambda$$

which holds for all values of $\mathbf{w}_1$ and $\mathbf{w}_2$, so ReLU is a convex function.

Question 6.13

Please see attached code for the cost history plots, and classification info.

Question 6.15

Please see attached code for the perceptron fit using the `scikit-learn` Python package. The confusion matrix is:

	Predicted	Values
—	Good	Bad
Good	576	124
Bad	133	167

Question 6.16

Although half of example 6.12 is cut off in the text by a picture, I was generally able to get the same accuracy. In the case where all weights are equal, the total accuracy of both classes is about 95% and accuracy of the red class is 60%. When the red class has a weight 5x higher than the others, the accuracy decreases to 93% but the red accuracy increases to 80%. In the case where the red weights are 10x higher, the total accuracy decreases to 91% but the red accuracy is now 100%.

HW 3

May 2, 2020

[1]:
```
#import numpy as np
import matplotlib.pyplot as plt
import autograd.numpy as np
from autograd import grad
from sklearn.linear_model import *
import sklearn.metrics as metrics
```

1 Problem 6.13

[2]:
```
def model(x, w):
    return w[0] + np.dot(x.T, w[1:])

def softmax(w):
    return np.sum(np.log(1 + np.exp(-y.flatten() * model(x, w)))) / np.size(y)

def perceptron(w):
    cost = 0
    xpTw = -y.flatten() * model(x, w)
    for i in range(xpTw.size):
        cost += np.max([0, xpTw[i]])
    return cost / np.size(y)

def classify(x, y, w):
    y_p = y.flatten()
    I = 0
    y_hat_p = np.sign(model(x, w))
    for i in range(y.size):
        if not np.isclose(y_hat_p[i], y_p[i]):
            I += 1
    return I
```

[3]:
```
data = np.loadtxt("breast_cancer_data.csv", delimiter=',')
x, y = data[:-1, :], data[-1:, :]
```

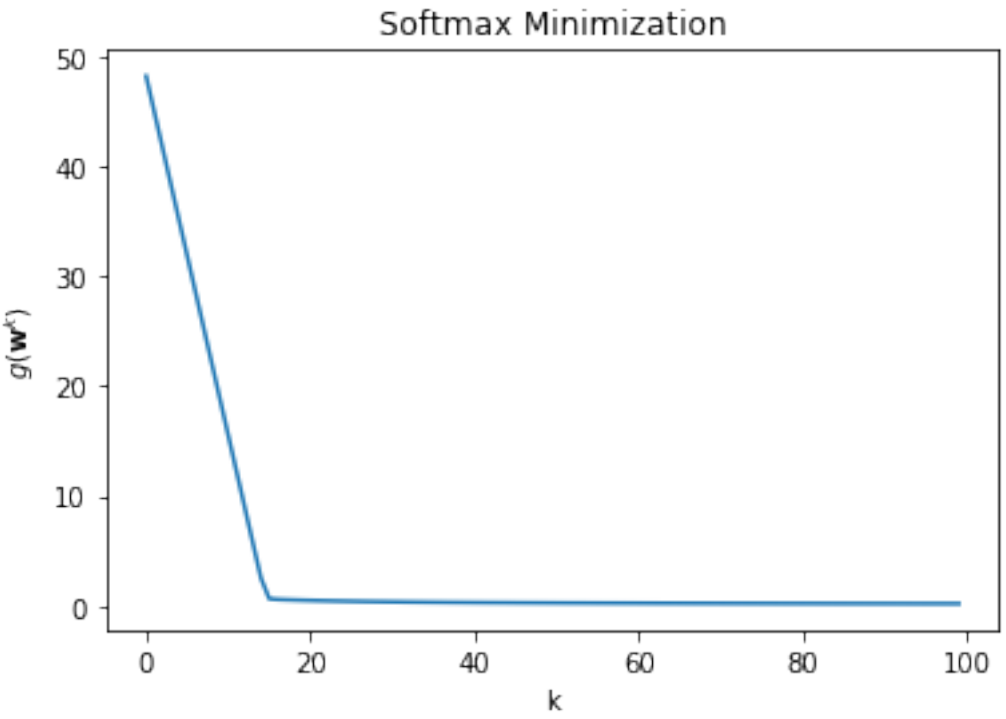
[4]:
```
w_soft = 3*np.ones(x.shape[0] + 1)
cost_soft = []
```

```
grad_softmax = grad(softmax)
for i in range(100):
    cost_soft.append(softmax(w_soft))
    w_soft -= 0.1 * grad_softmax(w_soft)

print("Number of mis-classifications: ", classify(x, y, w_soft))
print("Percent misclassified: %0.2f " % (100 * classify(x, y, w_soft) / y.size))
```

```
Number of mis-classifications:  45
Percent misclassified: 6.44
```

[5]:
```
plt.plot(cost_soft)
plt.xlabel("k")
plt.ylabel("$g(\\mathbf{w}^k)$")
plt.title("Softmax Minimization")
```

[5]: Text(0.5, 1.0, 'Softmax Minimization')



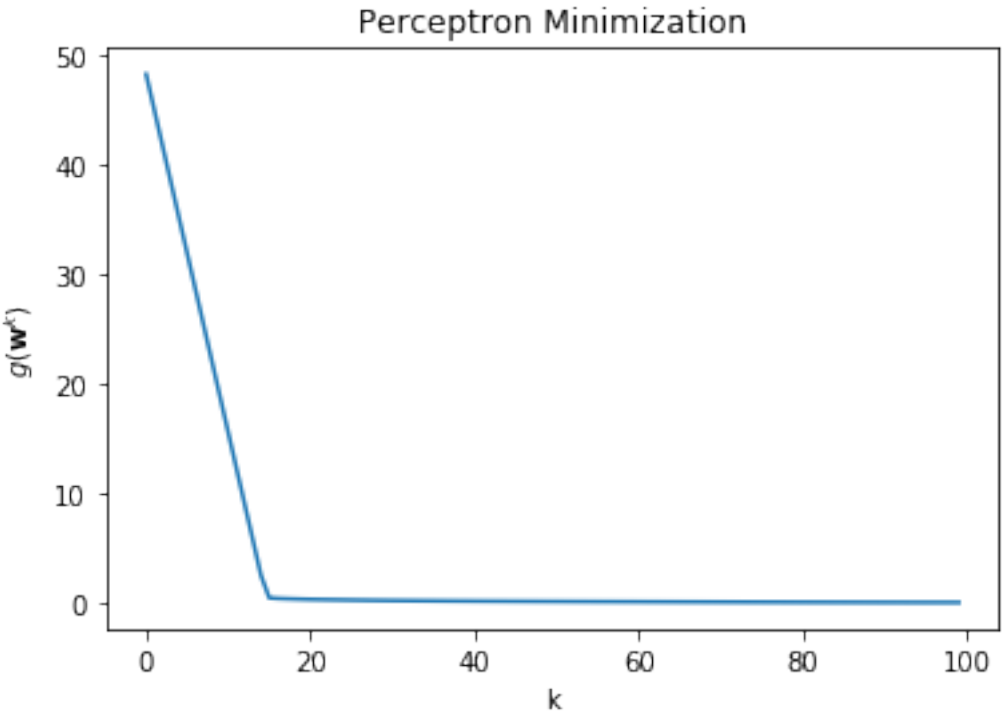
[6]:
```
w_per = 3*np.ones(x.shape[0] + 1)
cost_per = []
grad_percep = grad(perceptron)
for i in range(100):
    cost_per.append(perceptron(w_per))
    w_per -= 0.1 * grad_percep(w_per)
```

```
print("Number of mis-classifications: ", classify(x, y, w_per))
print("Percent misclassified: %0.2f " % (100 * classify(x, y, w_per) / y.size))
```

```
Number of mis-classifications:  33
Percent misclassified: 4.72
```

[7]:
```
plt.plot(cost_per)
plt.xlabel("k")
plt.ylabel("$g(\\mathbf{w}^k)$")
plt.title("Perceptron Minimization")
```

[7]: Text(0.5, 1.0, 'Perceptron Minimization')



2 Problem 6.15

[8]:
```
#Load and split the data
data = np.loadtxt("credit_dataset.csv", delimiter=',')

x, y = data[:-1, :], data[-1:, :]
print(x.shape, y.shape)

#Transform it via z-score
```

```
transformed = np.zeros(x.shape)

mean = np.nanmean(x, axis=1)
std = np.nanstd(x, axis=1)

for i in range(len(mean)):
    transformed[i, :] = (x[i, :] - mean[i]) / std[i]
```

```
(20, 1000) (1, 1000)
```

[9]:
```
#Initialize the Perceptron fitter and fit our data
clf = Perceptron(max_iter=100000, early_stopping=True, fit_intercept=True,␣
 ↪warm_start=True)
clf.fit(transformed.T, y.flatten())
```

[9]:
```
Perceptron(alpha=0.0001, class_weight=None, early_stopping=True, eta0=1.0,
           fit_intercept=True, max_iter=100000, n_iter_no_change=5, n_jobs=None,
           penalty=None, random_state=0, shuffle=True, tol=0.001,
           validation_fraction=0.1, verbose=0, warm_start=True)
```

[10]:
```
#Determine the predicted values
y_p_hat = clf.predict(transformed.T)
```

[11]:
```
#Get elements of confusion matrix
A, B, C, D = metrics.confusion_matrix(y.T, y_p_hat).flatten()
```

[12]:
```
#Print all the statistical metrics out using the confusion matrix
print("Accuracy of fit: %0.1f percent" % (100 * (A + D) / 1000))
print("----------------")
print("Good credit correctly predicted: ", D)
print("Good credit incorrectly predicted: ", C)
print("Good credit accuracy: %0.1f percent" % (100 * D / (C + D)))
print("----------------")
print("Bad credit correctly predicted: ", A)
print("Bad credit incorrectly predicted: ", B)
print("Bad credit accuracy: %0.1f percent" % (100 * A / (A + B)))
```

```
Accuracy of fit: 74.3 percent
----------------
Good credit correctly predicted:  576
Good credit incorrectly predicted:  124
Good credit accuracy: 82.3 percent
----------------
Bad credit correctly predicted:  167
Bad credit incorrectly predicted:  133
Bad credit accuracy: 55.7 percent
```

3 Problem 6.16

[13]:
```
data = np.loadtxt("3d_classification_data_v2_mbalanced.csv", delimiter=',')
x, y = data[:-1, :], data[-1:, :].flatten()

print(x.shape, y.shape)
```

```
(2, 55) (55,)
```

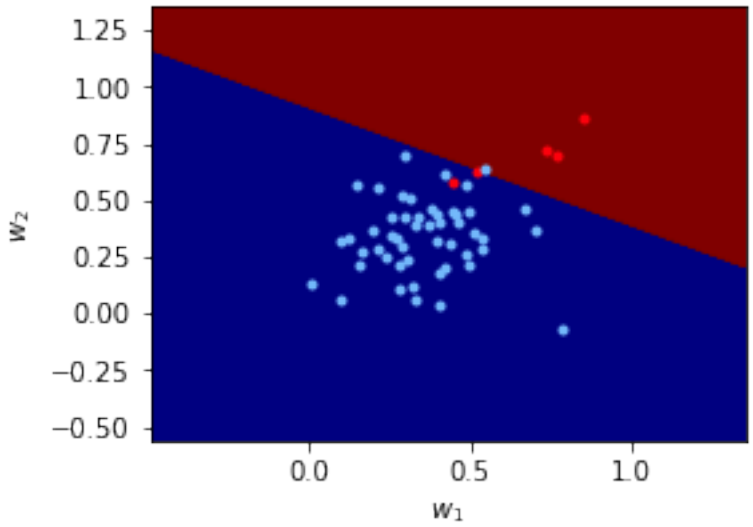
[14]:
```
#Softmax regression with no penalty
clf = LogisticRegression(multi_class='multinomial', solver='newton-cg',␣
 ↪penalty='none')
```

[15]:
```
#Equal weights and fit the data
beta = np.ones(y.shape)
clf.fit(x.T, y, sample_weight=beta)

#Determine the accuracy
y_hat_p = clf.predict(x.T)
A, B, C, D = metrics.confusion_matrix(y.T, y_hat_p).flatten()
print("Accuracy: %0.1f percent" % (100 * (A + D) / (A + B + C + D)))
```

```
Accuracy: 94.5 percent
```

[16]:
```
x_min, x_max = x[0, :].min() - .5, x[0, :].max() + .5
y_min, y_max = x[1, :].min() - .5, x[1, :].max() + .5

h = .001  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap='jet', alpha=0.1)
plt.plot(x[0, y==1], x[1, y==1], '.', color='xkcd:bright red')
plt.plot(x[0, y==-1], x[1, y==-1], '.', color='xkcd:sky blue')
plt.ylabel("$w_2$")
plt.xlabel("$w_1$")
```

[16]:
```
Text(0.5, 0, '$w_1$')
```

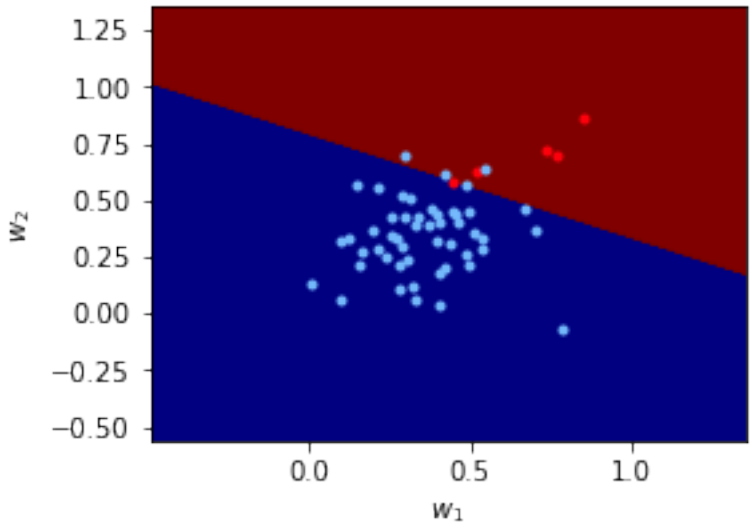


[17]:
```
#Red weights 5x higher than others
beta = np.ones(y.shape)
beta[y == 1] = 5.0
clf.fit(x.T, y, sample_weight=beta)

#Determine the accuracy
y_hat_p = clf.predict(x.T)
A, B, C, D = metrics.confusion_matrix(y.T, y_hat_p).flatten()
print("Accuracy: %0.1f percent" % (100 * (A + D) / (A + B + C + D)))
```

```
Accuracy: 92.7 percent
```

[18]:
```
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap='jet', alpha=0.1)
plt.plot(x[0, y==1], x[1, y==1], '.', color='xkcd:bright red')
plt.plot(x[0, y==-1], x[1, y==-1], '.', color='xkcd:sky blue')
plt.ylabel("$w_2$")
plt.xlabel("$w_1$")
```

[18]:
```
Text(0.5, 0, '$w_1$')
```

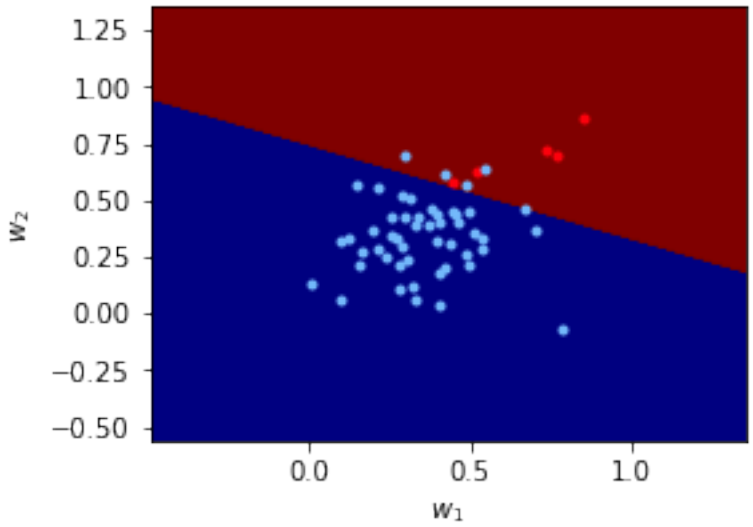


[19]:
```
#Red weights 10x higher than others
beta = np.ones(y.shape)
beta[y == 1] = 10.0
clf.fit(x.T, y, sample_weight=beta)

#Get the accuracy
y_hat_p = clf.predict(x.T)
A, B, C, D = metrics.confusion_matrix(y.T, y_hat_p).flatten()
print("Accuracy: %0.1f percent" % (100 * (A + D) / (A + B + C + D)))
```

```
Accuracy: 90.9 percent
```

[20]:
```
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap='jet', alpha=0.1)
plt.plot(x[0, y==1], x[1, y==1], '.', color='xkcd:bright red')
plt.plot(x[0, y==-1], x[1, y==-1], '.', color='xkcd:sky blue')
plt.ylabel("$w_2$")
plt.xlabel("$w_1$")
```

[20]:
```
Text(0.5, 0, '$w_1$')
```


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
w_2
0.0
0.5
1.0
w_1

[ ]: