COGS 181 - Homework 3

1 - Perceptron

1.1 Programming

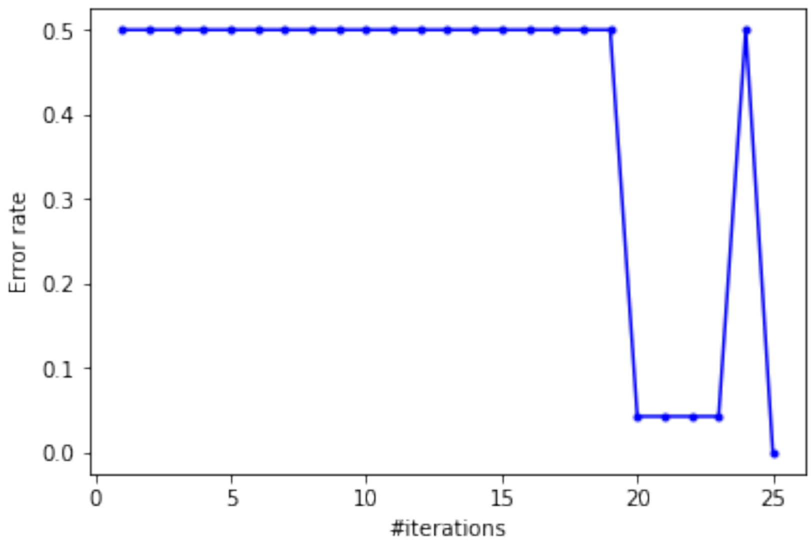
```
import numpy as np
import pandas as pd
# Read data
data = pd.read_csv('Q1_data.txt',header=None, usecols=[0,1,2,3,4])

# Split up X and Y
X = data.as_matrix()[:,:4]
Y = data.as_matrix()[:,-1]

# Find index for all the different classes
mask_setosa = np.where(Y == "Iris-setosa")[0]
mask_vc = np.where(Y == "Iris-versicolor")[0]

# Classify to 1 and -1 for the two classes
Y[mask_setosa] = 1
Y[mask_vc] = -1

# Find index for wanted training and testing tests
training_mask = np.append(mask_setosa[0:35], mask_vc[0:35])
testing_mask = np.append(mask_setosa[35::], mask_vc[35::])

# Split X and Y to training & testing sets
X_train = X[training_mask]
X_test = X[testing_mask]
Y_train = Y[training_mask]
Y_test = Y[testing_mask]
```

```
import random
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
%matplotlib inline

# Returns the predicted values of our classifier
# X is Nx4
# w is 4x1
```

```python
# b is scalar
def f(X,w,b):
    # Our classifier
    values = np.dot(X,w) + b
    # Map all calculated values to -1 or 1 given value >= 0
    mapper = lambda x: [-1,1][x >= 0]
    vectorizer = np.vectorize(mapper)
    predictions = vectorizer(values)
    return predictions

def error(X,Y,w,b):
    predictions = f(X,w,b)
    # error can also be calculated manually by
    correct = predictions == Y
    accuracy = sum(correct) / float(len(X))
    error = 1 - accuracy
    # Can be calulated with sklearn function as well...
    # error = 1- accuracy_score(Y.astype(int), predictions)
    return error

# X Nx4
# Y Nx1
# w 4x1
# b scalar
def perceptron_learn(X, Y, w, b, max_iterations):
    ws = []
    bs = []
    error_rate = 1
    it = 0
    errors = []
    while error_rate > 0 and it < max_iterations:
        error_rate = error(X,Y,w,b)
        errors.append(error_rate)
        it = it + 1
        ws.append(w)
        bs.append(b)
        # Select a random point
        i = random.randrange(0, len(X))
        # Calculate prediction
        prediction = f(X[i], w, b)
        if prediction == Y[i]:
            continue
        else:
            w = w + lam*(Y[i] - prediction)*X[i]
            b = b + lam*(Y[i] - prediction)

    return ws,bs,errors
```

```
w = np.array([random.random() for i in range(len(X[0]))])
b = random.random()
lam = 0.1

ws ,bs ,e = perceptron_learn(X_train,Y_train, w, b, 1000)
print "Perceptron learn done"
w = ws[-1]
b = bs[-1]

plt.plot([x for x in range(1,len(e)+1)], e, "-b.", label="Error rate")
plt.xlabel("#iterations")
plt.ylabel("Error rate")
print "Trained W:\n", w
print "Trained b:", b
```

```
Perceptron learn done
Trained W:
[0.31867003734174293 1.581169284491959 -2.0040990369799783
 -0.7228558941526755]
Trained b: 0.415828142284
```



1.2 - Decision boundary

The decision boundary is given by

$$w^T x + b = 0$$

We can find the points of the hyperplane:

1.2

One of the solutions are

$w = [-0.02, 0.45, -0.77, 0.4]$, $b = 0.48$

$w^T x + b \geq 0$

$-0.02x_0 + 0.45x_1 - 0.77x_2 + 0.4x_3 + 0.48 = 0$

setting $x_1, x_2, x_3 = 0$ gives

$-0.02x_0 + 0.48 = 0 \Rightarrow$

$\underline{x_0 = 24}$

$x_0, x_2, x_3 = 0$

$\underline{x_1 = -1{,}07}$

$x_0, x_1, x_3 = 0$

$\underline{x_2 = 0{,}62}$

$x_0, x_1, x_2 = 0$

$\underline{x_3 = -1{,}2}$

This defines the decision boundry as an hyperplane given by the points x_0, x_1, x_2 and x_3.

1.3 - Test

```python
# #correct_predictions / #predictions
def get_accuracy(Y, predictions):
    assert Y.shape == predictions.shape
    correct_predictions = np.equal(Y, predictions)
    correct = sum(correct_predictions)

    return correct / float(len(predictions))

# #true_positives / (#true_positives + #false_positives)
def get_precision(Y, predictions):
    assert Y.shape == predictions.shape
    true_positives = get_true_positives(Y, predictions)
    all_positives =  (predictions == 1).sum()
    if all_positives == 0:
        return 1
    return true_positives / float(all_positives)

def get_true_positives(Y, predictions):
    TP = np.logical_and(predictions == 1, Y == 1).sum()
    return TP

# #true_positives / #positives_in_test_set
def get_recall(Y, predictions):
    assert Y.shape == predictions.shape
    TP = get_true_positives(Y, predictions)
    positives_in_Y = (Y==1).sum()
    return TP / float(positives_in_Y)

def get_f_value(precision, recall):
    return 2 * precision * recall / float(precision + recall)

def print_stats(Y, predictions):
    accuracy = get_accuracy(Y, predictions)
    precision =  get_precision(Y, predictions)
    recall = get_recall(Y, predictions)
    f_value = get_f_value(precision, recall)
    print "Accuracy:", accuracy
    print "Precision:", precision
    print "Recall:", recall
    print "F Value:", f_value


predictions = f(X_test, w, b)
print_stats(Y_test, predictions)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F Value: 1.0
```

2 - Logistic Regression 1

1

2 a) $$P(Y=0|x) = 1 - P(Y=1|x) = 1 - \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} = \frac{1+e^{\alpha+\beta x} - e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

$$P(Y=0|x) = \frac{1}{1+e^{\alpha+\beta x}}$$

$$P(Y=0|X) = \frac{1}{1+e^{\alpha+\beta x}}$$

2

Proof is done case-by-case for all possible values of y.

b) $$[P(Y=1|x)]^{Y} \times [P(Y=0|x)]^{1-Y} = \frac{1}{1+e^{-(2y-1)(\alpha+\beta x)}}$$

We show it case by case for $y \in (0,1)$

$$\left(\frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}\right)^{Y} \left(\frac{1}{1+e^{\alpha+\beta x}}\right)^{1-Y} = \frac{1}{1+e^{-(2y-1)(\alpha+\beta x)}}$$

$Y=1$

LHS

$$\frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} = \frac{1}{1+e^{-(\alpha+\beta x)}}$$

RHS

$$\frac{1}{1+e^{-(2-1)(\alpha+\beta x)}} = \frac{1}{1+e^{-(\alpha+\beta x)}}$$

$Y=0$

LHS

$$\left(\frac{1}{1+e^{\alpha+\beta x}}\right)^{1-0}$$

RHS

$$\frac{1}{1+e^{-(0-1)(\alpha+\beta x)}} = \frac{1}{1+e^{(\alpha+\beta x)}}$$

We see that it is correct for all possible values of y.

3

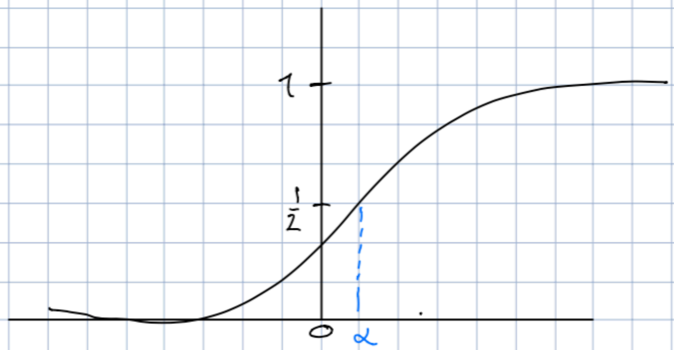
The decision boundary is given by

$$\alpha + \beta x = 0$$

3) The decision boundry is at $\alpha + \beta x = 0$

$$P(y=1 \mid x) = \frac{1}{1+e^{-(\alpha+\beta x)}}$$

It is related to the sigmoid as it's the same decision boundry.



It is also the logit of the sigmoid.

3 - Logistic Regression 2

1 - Derivation

$$L(\omega) = -\sum_i y_i \ln p_i + (1-y_i)\ln(1-p_i)$$

$$\frac{dL}{d\omega} = -\sum \frac{y_i}{p_i}\frac{dp_i}{d\omega} + \frac{1-y_i}{1-p_i}\left(-\frac{dp_i}{d\omega}\right) = -\sum \frac{dp_i}{d\omega}\left(\frac{y_i}{p_i} - \frac{1-y_i}{1-p_i}\right)$$

$$\boxed{\frac{dL}{d\omega} = \sum \frac{dP_i}{d\omega}\left(\frac{P_i - y_i}{P_i(1-P_i)}\right)}$$

$$\frac{dP_i}{d\omega} = \frac{X_i e^{-(\omega^T x_i + b)}}{(1+e^{-(\omega^T x_i + b)})^2} = X_i P_i^2 e^{-(\omega^T x_i + b)}$$

$$\frac{dL}{d\omega} = \sum_i X_i e^{-(\omega^T x_i + b)} \cdot P_i^2 \left(\frac{P_i - y_i}{P_i(1-P_i)}\right) = \sum_i X_i e^{-(\omega^T x_i + b)} \cdot P_i \frac{\frac{1}{1+e^{-}} - y_i}{\underbrace{1 - \frac{1}{1+e^{-}}}_{= \frac{e^{-}}{1+e^{-}}}}$$

$$\sum_i X_i \cancel{e^{-(\omega^T x_i + b)}} \cdot \cancel{P_i} \frac{P_i - y_i}{\cancel{e^{-}}\,\cancel{P_i}} = \sum_i X_i (P_i - Y_i)$$

$$\boxed{\frac{dL}{d\omega} = \sum_i X_i (P_i - y_i)}$$

$$\frac{dL}{d\omega} = \sum_i X_i (P_i - y_i)$$

2 - Update rule of ω

2) The update rule for ω is

$$\omega_{t+1} = \omega_t - \alpha \nabla f(\omega_t) = \omega_t - \alpha \frac{dL}{d\omega}$$

$$\boxed{\omega_{t+1} = \omega_t - \alpha \sum_i X_i (P_i - y_i)}$$

Update rule of b

$$\frac{dL}{db} = \sum_i \left(\frac{dP_i}{db} \left(\frac{P_i - y_i}{P_i(1-P_i)} \right) \right)$$

$$\frac{dP_i}{db} = \frac{e^{-(w^T x_i + b)}}{(1 + e^{-(w^T x_i + b)})^2} = P_i^2 e^{-(w^T x_i + b)}$$ This gives

$$\boxed{\frac{dL}{db} = \sum_i (P_i - y_i)}$$

$$\boxed{b_{t+1} = b_t - \alpha \sum_i (P_i - y_i)}$$

3.2 - Logistic Regression implementation

Data reading

```
data = np.loadtxt('Q3_data.txt',
                  delimiter=',',
                  converters={-1: lambda s: {b'Iris-versicolor': 0,
                                             b'Iris-virginica': 1}[s]
                  })
Y = data[:,4]
X = data[:, [0,1,2,3]]


mask_0 = np.where(Y == 0)[0]
mask_1 = np.where(Y == 1)[0]

# Find index for wanted training and testing tests
testing_mask = np.append(mask_setosa[0:15], mask_vc[0:15])
training_mask = np.append(mask_setosa[15::], mask_vc[15::])

# Split X and Y to training & testing sets
X_train = X[training_mask]
X_test = X[testing_mask]
Y_train = Y[training_mask]
Y_test = Y[testing_mask]
```

Implementation

```
from math import exp, log
from matplotlib.legend_handler import HandlerLine2D
```

```
# Sigmoid function
# x_i = 4x1
# w   = 4x1
# b   = scalar
def P(x_i, w, b):
    assert x_i.shape == (4,), "X shape has to be 4x1. Was {}, {}".format(x_i.shape,
    assert w.shape == (4,), "W shape has to be 4x1"

    upper = np.dot(w, x_i.T) + b
    res = 1 / float(( 1 + exp(-upper)))
    return res

def predict(X,w,b):
    return np.asarray([[0,1][P(x_i, w, b) >= 0.5] for x_i in X])


# Loss function
def loss(Y, X, w, b):
    assert X[0].shape == (4,), "X shape is supposed to be (4,). it was: {0}".format(
    res = 0
    for i in range(len(X)):
        p_i = P(X[i], w, b)
        res += Y[i] * log(p_i) + (1 - Y[i]) * log(1 - p_i + 0.000000001)
    return res

def gradient_descent(X, Y, w ,b, alpha, max_iterations):
    # Validate parameters
    m = len(X)
    n = len(w)
    assert X.shape == (m,n)
    assert Y.shape == (m,)
    assert w.shape == (n,)

    # Variables to track w's, b's, losses
    it = 0
    losses = []
    ws = []
    bs = []
    for i in range(max_iterations):
        # Calulate predict values
        y_predict = predict(X, w, b)

        # Find dw, db
        dw = np.dot(X.T, y_predict - Y )
        db = float((y_predict - Y).sum())
        assert dw.shape == (n,)

        # Update w, b
```

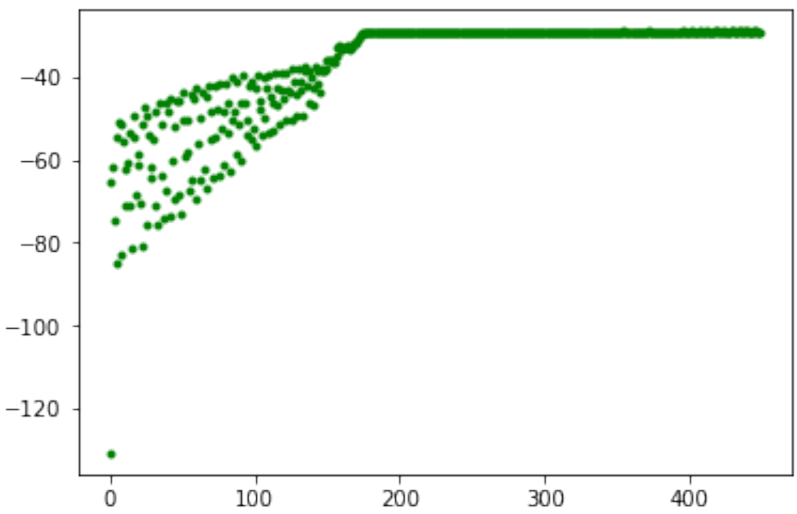
```
        w = w - alpha * dw
        b = b - alpha * db

        # Record values
        losses.append(loss(Y, X, w, b))
        ws.append(w)
        bs.append(b)
        it += 1
    return losses, ws, bs

w = np.array([random.random() for x in range(len(X_train[0]))])
b = random.random()
losses, ws, bs = gradient_descent(X_train, Y_train, w, b, 0.001, 450)
print "W:", ws[-1]
print "B:", bs[-1]
```

Output

```
W: [-1.08229593 -0.37491066  1.2339429   0.92939273]
B: 0.203415225806
```

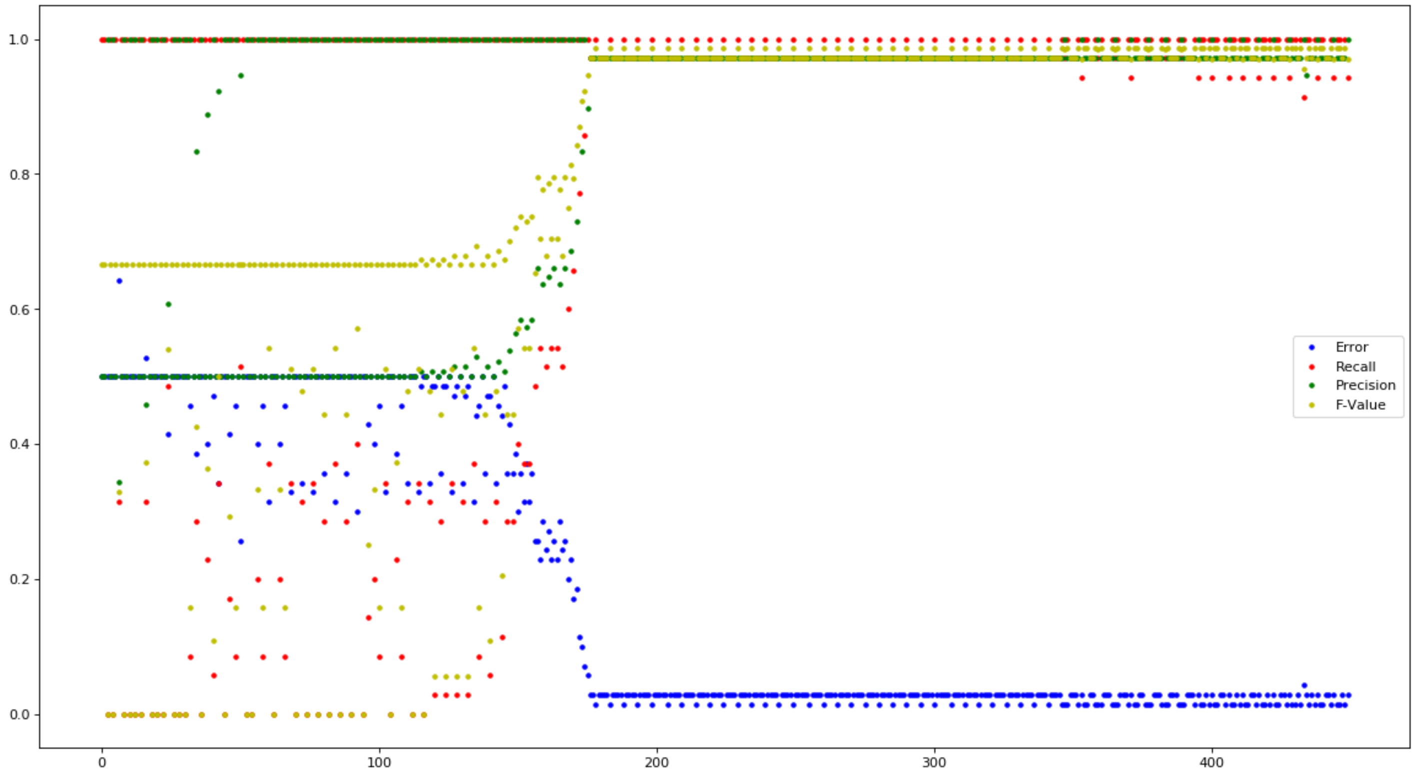
```
plt.plot(range(0,len(losses)), losses, "g.")
print "Final loss:", losses[-1]
```

```
Final loss: -29.3565085786
```



Statistics from training

```
plot_statistics(Y_train,X_train, predict, ws ,bs)
```



3.3 - Decision Boundary

The decision boundary is given from p_i when it is below / above 0.5

That is

$$p_i \geq 0.5$$

This is when

$$w^T X_i + b = 0$$

If we plug in the values of w from the training into this equation, we can derive the decision boundary as a hyperplane

3.4 - Test

```
predictions = predict(X_test, ws[-1], bs[-1])
accuracy = get_accuracy(Y_test, predictions)
precision =  get_precision(Y_test, predictions)
recall = get_recall(Y_test, predictions)
f_value = get_f_value(precision, recall)
print "Accuracy:", accuracy
print "Precision:", precision
print "Recall:", recall
print "F Value:", f_value
```

```
Accuracy: 0.9
Precision: 0.833333333333
Recall: 1.0
F Value: 0.909090909091
```

4 - Logistic Regression 3

1

4

$P(Y=1|X) = \frac{1}{1+e^{-(w^Tx+b)}}$, $P(Y=-1|X) = \frac{1}{1+e^{w^Tx+b}}$

Show that $P(Y|X) = \frac{1}{1+e^{-y(w^Tx+b)}}$ when $y \in \{-1, 1\}$

I'll show it case-by-case

$Y=1$

$$P(Y=1|X) = \frac{1}{1+e^{-(w^Tx+b)}}$$

$$P(Y|X) = \frac{1}{1+e^{-1(w^Tx+b)}}$$

$Y=-1$

$$P(Y=-1|X) = \frac{1}{1+e^{(w^Tx+b)}}$$

$$P(Y|X) = \frac{1}{1+e^{-(-1)(w^Tx+b)}}$$

We see that $P(Y|X)$ is correct for $y \in \{-1, 1\}$

2

2) It will classify y as 1 when $\tilde{P}(Y|x) \geq 0.5$ and -1 otherwise. This boundary is at $w^T x + b = 0$.

The decision boundary is at

$$w^T x + b = 0$$