CS-722 Machine Learning

Homework #3

Joseph S. Cannella

1	100
-3.702	-7.40
-1.935	-3.88

Homework #3

1.) Given: Logistic Loss: $\vec{\mathcal{L}}(\vec{w}) = \sum_{i=1}^{N} \ln(1+\exp(\vec{X}_i^T\vec{w})) - Y_i\vec{X}_i^T\vec{w}$

Find: $\frac{\delta \mathcal{L}}{\delta \vec{w}} = ?$

let $\mathcal{L}(\vec{w}) = \sum_{i=1}^{N} l_i(\vec{w})$

Formulas:

$\frac{d}{dx}\ln(f(x)) = \frac{1}{f(x)} f'(x)$

$\frac{d}{dx} e^{ax} = ae^{x}$; $\frac{\delta x^T a}{\delta x} = \frac{\delta a^T x}{\delta x} = a$

$l_i(\vec{w}) = \ln(1+\exp(\vec{X}_i^T\vec{w})) - Y_i\vec{X}_i^T\vec{w}$

$$\frac{\delta l_i}{\delta w} = \underbrace{\frac{\exp(\vec{X}_i^T\vec{w})}{1+\exp(\vec{X}_i^T\vec{w})}}_{\text{sigmoid} \to \sigma(\vec{X}_i^T\vec{w}) \leftarrow \text{"logit"}} \vec{X}_i - \vec{X}_iY_i = \underbrace{\underbrace{\left\{ \overbrace{\frac{\exp(\vec{X}_i^T\vec{w})}{1+\exp(\vec{X}_i^T\vec{w})}}^{\hat{Y}_i} - Y_i \right\}}_{\text{scalar}} \cdot \underbrace{\vec{X}_i}_{\text{vector}}}_{\text{vector}}$$

$$\boxed{\frac{\delta \mathcal{L}(\vec{w}_j)}{\delta \vec{w}_j} = \sum_{i=1}^{N} \left\{ \frac{\exp(\vec{X}_i^T\vec{w})}{1+\exp(\vec{X}_i^T\vec{w})} - Y^{(i)} \right\} \vec{X}_j^{(i)}}$$

$a \cdot b =$

let X_i be row of ith input

$\vec{X}_i^T\vec{w} = X_i\vec{w} \to (1,30)\cdot(30,1)$

$$\boxed{\nabla_{\vec{w}}\mathcal{L}(\vec{w}) = \underline{X}^T \cdot \{\sigma(\underline{X}\cdot\vec{w}) - \vec{Y}\}}$$

```
# 2.) Program your own logistic regression classifier (Python is preferred) by im
plementing a
# gradient decent algorithm to find the optimal � that minimizes the logistic lo
ss
#%%
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt

# X = [|,|,|], x_i = |
# w = [s,s,s]
# Y = |

# logit is a type of sigmoid function
def logit(x):
    # np.exp(x)/(1+np.exp(x)) -> Make it easier for computer
    # 1.0/(1+np.exp(-x))
    return np.divide(1.0, (1.0+np.exp(-x)))

def log_probability(X, w_vect):
    # Calculate weighted sum of X
    weighted_sum = X.dot(w_vect)
    return logit(weighted_sum)

# Vectorized Gradient for Logistic Regression
def log_gradient(X, Y, w_vect):
    return (1.0/X.shape[0])*X.T.dot((np.subtract(Y, logit(X.dot(w_vect)))))

# Algorithm of gradient descent
"""
1. Set iteration ! = 0, make an initial guess $%
2. repeat:
3. Compute the negative gradient of E(w) at w_k
and set it to be the search direction d_k
4. Choose a step size alpha_k to sufficiently reduce E(w_k + alpha_k*d_k)
5. Update w_(k+1) = w_k + alpha_k+d_k
6. k = k + 1
7. Until a termination rule is met
"""

def gradient_descent(X, Y, w, itterations=100, alpha=0.01):
    print('Original shape of w')
    print(w.shape)
```

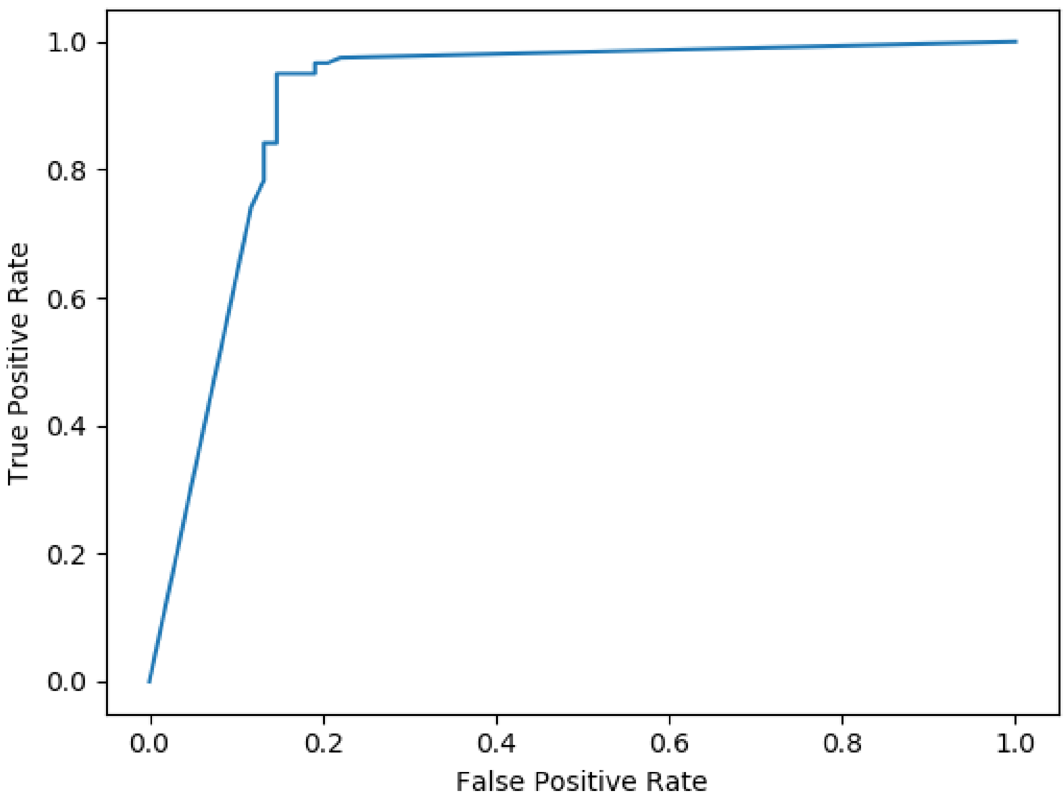
```python
    for i in range(0, itterations):
        # Use negative gradient to detemine search direction for minima
        d = log_gradient(X, Y, w)
        # Re-compute weighting parametric w
        w = w + (alpha*d)
        # Reduce step size
        alpha = alpha/2.0
    return w


# 3.)
"""
Program (with python preferred) a function that plots a ROC curve with inputs of
a
vector containing the true label and another vector containing the predicted prob
abilities
of class membership for a set of examples.
"""

#%%
def calculate_confusion_matrix(actuals, P_pred, threshold):
    tp=tn=fp=fn=0
    for actual, prob in zip(actuals, P_pred):

        # Predicted True
        if prob > threshold:
            if actual == 1:
                tp += 1
            else:
                fp += 1
        # Predicted False
        else:
            if actual == 0:
                tn += 1
            else:
                fn += 1

    return (tp, fp, tn, fn)

def FPR(fp, tn):
    return fp/(fp+tn)

def TPR(tp, fn):
    return tp/(tp+fn)
```

```
def get_all_TPR_FPR(actual, P_pred):
    P_min = min(P_pred)
    P_max = max(P_pred)
    stepSize = (abs(P_max) + abs(P_min))/1000

    thresholds = np.arange(P_min-stepSize, P_max+stepSize, stepSize)

    FPRs = []
    TPRs = []

    for thresh in thresholds:
        tp, fp, tn, fn = calculate_confusion_matrix(actual, P_pred, thresh)
        TPRs.append(TPR(tp, fn))
        FPRs.append(FPR(fp, tn))

    return TPRs, FPRs


def ROC(actuals, P_pred):
    TPRs, FPRs = get_all_TPR_FPR(actuals, Y_prob)

    plt.plot(FPRs, TPRs)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    return TPRs, FPRs


#%%
"""4.) Apply your logistic regression classifier to the breast cancer Wisconsin d
ataset, which
can either be loaded with python by following instructions here:
http://scikit-
learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
or downloaded from
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) .
Randomly splitting the data into two subsets with one having 2/3 of the examples
and the other
having the rest 1/3. Use the 2/3 subset to train a logistic regression model and
the 1/3 subset to
test the model. Plot the ROC curve on the testing set with your ROC plotting func
tion.
ROC(Y, Y_prob)"""

# Extract the relevant data from the source
```

```
# Test logistic regression classifier
data = load_breast_cancer()
X = data.data
Y = data.target.reshape(X.shape[0], 1)

# Split data into Train and Test Sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)

# %%

# Create initial guess for Paramaterized Weights
w = np.zeros(X.shape[1])
w = w.reshape(X.shape[1], 1)

# Calculate the optimal values for this problem
w_opt = gradient_descent(X_train, Y_train, w)
print("Optimal Values for w:")
print(w_opt.reshape([1, w_opt.shape[0]]))

# Calculate the Predicted Probabailites of Postitive Results
Y_prob = log_probability(X_test, w_opt)
Y_pred = np.around(Y_prob)

print("Prediction Accuracy")
print(accuracy_score(Y_test.flatten(),Y_pred.flatten()))

# Display ROC Curve
TPRs, FPRs = ROC(Y_test, Y_prob)
```

Output:

```
Original shape of w
(30, 1)
Optimal Values for w:
[[ 3.39659099e-02  6.39270779e-02  2.08407221e-01  3.42143322e-01
   3.50720393e-04  1.23350360e-04 -1.79558620e-04 -8.96976837e-05
   6.59035538e-04  2.57438682e-04  1.28346771e-04  5.03255770e-03
   6.64179716e-04 -7.83209466e-02  3.04122341e-05  5.00196325e-05
   5.07846735e-05  2.35424577e-05  8.28667116e-05  1.22681084e-05
   3.15874138e-02  8.08953757e-02  1.92400815e-01 -3.23297155e-01
   4.57389927e-04  1.33420444e-04 -2.15285873e-04 -3.15164000e-05
   9.43918265e-04  2.84734388e-04]]
Prediction Accuracy
0.8936170212765957
```


True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0