(2)

a) Why logistic loss function does not suffer from the same problem as the squared error loss on easy to classify points

$f(w)$ = squared error = $\|Aw - \alpha\|_2^2 = (Aw-\alpha)^T(Aw-\alpha) = w^TA^TAw - 2w^TA^T\alpha + \alpha^T\alpha$

$f'(w) = \nabla_w f(w) = 2A^TAw - 2A^T\alpha = 2A^T(Aw-\alpha)$

$f''(w) = 2A^TA$

- Then, for misclassifications the squared error function does not strongly penalize them. Also, the least square is not a convex function.

$g(w)$ = logistic loss function = $\log(1+e^{-\alpha A^T w})$

$$g'(w) = \frac{-\alpha A^T}{1+e^{-\alpha A^T w}}$$

$$g''(w) = \frac{\alpha A^T \alpha A^T e^{-\alpha A^T w}}{(1+e^{-\alpha A^T w})^2}$$

- The logistic function heavily penalizes misclassifications because of the logaritmic function.
  Also, $e^x \in (0, +\infty)$ so $g''(w)$ is always $\geq 0 \Rightarrow$ the function is convex.

wikipedia_wisconsin_starter

April 11, 2022

[8]:
```
import numpy as np
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import eigs

edges_file = open('wisconsin_edges.csv', "r")
nodes_file = open('wisconsin_nodes.csv', "r")

# create a dictionary where nodes_dict[i] = name of wikipedia page
nodes_dict = {}
for line in nodes_file:
    nodes_dict[int(line.split(',',1)[0].strip())] = line.split(',',1)[1].strip()

node_count = len(nodes_dict)

# create adjacency matrix
A = np.zeros((node_count, node_count))
for line in edges_file:
    from_node = int(line.split(',')[0].strip())
    to_node = int(line.split(',')[1].strip())
    A[to_node, from_node] = 1.0

## Add code below to (1) prevent traps and (2) find the most important pages ␣
↪
# Hint -- instead of computing the entire eigen-decomposition of a matrix X␣
↪using
# s, E = np.linalg.eig(A)
# you can compute just the first eigenvector with:
# s, E = eigs(csc_matrix(A), k = 1)
```

[9]:
```
for i in range(len(A)):
    for j in range(len(A)):
        A[i][j] += 0.001
A
```

[9]:
```
array([[0.001, 0.001, 0.001, …, 0.001, 0.001, 0.001],
       [0.001, 0.001, 0.001, …, 0.001, 0.001, 0.001],
       [0.001, 0.001, 0.001, …, 0.001, 0.001, 0.001],
```

```
        …,
        [0.001, 0.001, 0.001, …, 0.001, 0.001, 0.001],
        [0.001, 0.001, 0.001, …, 0.001, 0.001, 0.001],
        [0.001, 0.001, 0.001, …, 0.001, 0.001, 0.001]])
```

[13]:
```
A_norm = A/A.sum(axis=0, keepdims=1)
```

[15]:
```
s,E = eigs(csc_matrix(A_norm),k=1)
s,E
```

[15]:
```
(array([1.+0.j]),
 array([[-0.00094793+0.j],
        [-0.00113526+0.j],
        [-0.00094793+0.j],
        …,
        [-0.01864669+0.j],
        [-0.00164875+0.j],
        [-0.00094793+0.j]]))
```

[16]:
```
E = abs(E)
```

[20]:
```
e_dict = {}

for i in range(len(E)):
    e_dict[i] = E[i]

e_dict_sorted = sorted(e_dict.items(), key=lambda x : x[1], reverse=True)
print("B: " , e_dict_sorted[0][0])
print("C: ", e_dict_sorted[2][0])
```

```
B:  5089
C:  1345
```

classifier_starter

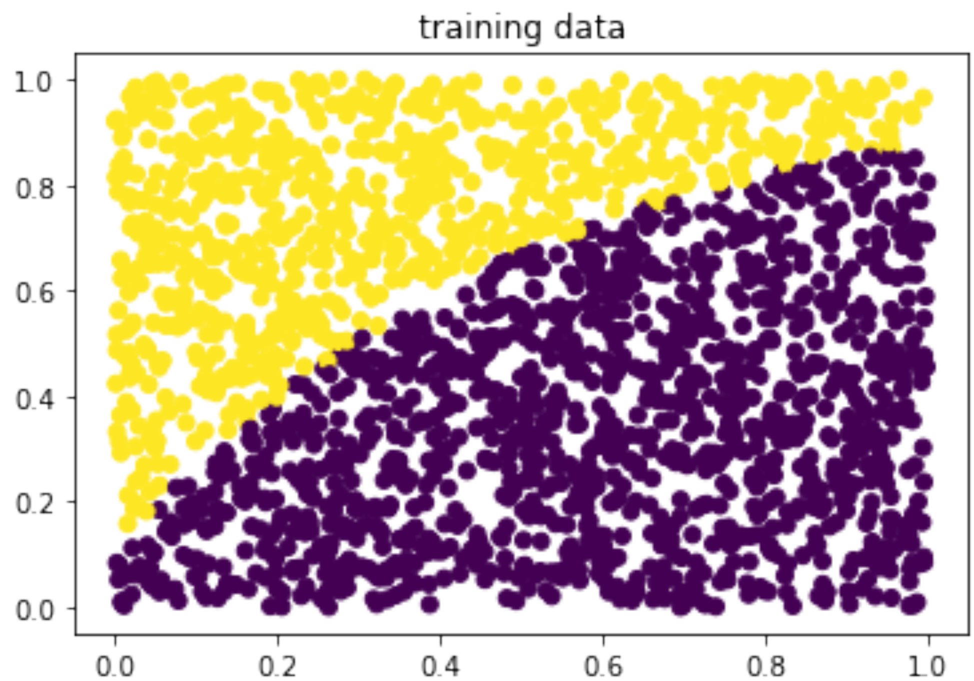
April 11, 2022

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pickle

pkl_file = open('classifier_data.pkl', 'rb')
x_train, y_train = pickle.load(pkl_file)

n_train = np.size(y_train)

plt.scatter(x_train[:,0],x_train[:,1], c=y_train[:,0])
plt.title('training data')
plt.show()
```



[6]:

```
def gradient(w,l):
    return np.sum([-1 * y_train.T @ x_train / (1 + np.exp(-1 * y_train.T @␣
 ↪x_train @ w))]) + 2 * l * w

def gradient_descent(starting_w,l):
    w_current = starting_w
    tau = (2/(np.linalg.norm(x_train, 2)**2))/2
    for i in range(100):
        w_current += -1.0 * tau * gradient(w_current,l)

    return w_current
```

[33]:

```
gradient_descent(np.array([[.5],[.5]],float),1)
```

[33]:

```
array([[0.41970883],
       [0.41970883]])
```

[23]:

```
tau = (2/(np.linalg.norm(x_train, 2)**2))/2
```