ML: HW4

1. Let's look at logistic loss

$$\ell_{logistic}(y,w) = \log(1+e^{-yw^Tx})$$

$$= \log(1/\sigma(yw^Tx))$$

where $\sigma(z) = \dfrac{1}{1+e^{-z}}$ is the sigmoid function.

Minimizing $\ell_{logistic}(y, w_1)$, we get

$$\nabla_w \ell_{logistic}(y,w_1) = \frac{d}{dw}\log(\sigma(yw_1^Tx)^{-1})$$

Let $z = yw_1^Tx \longrightarrow$ ①

$$\therefore \nabla_w \ell_{logistic}(y,w_1) = \sigma(z).\sigma(z).(1-\sigma(z)).\frac{dz}{dw_1}$$

$$= \sigma^2(z)(1-\sigma(z)).yx \quad \text{From ①}$$

Equating to zero

$$\Rightarrow \nabla_w l_{logistic}(y,w) = \sigma^2(z)(1-\sigma(z)).yx = 0$$

$$\Rightarrow \boxed{\sigma(y w_1^T x) = 1} \longrightarrow 2$$

Now looking at $p(y=1|x;w_2) = \frac{1}{1+e^{-x^T w_2}}$

We get negative log-likelihood as

$$NLL_D(w_2) = \sum_{i=1}^{n} \log\left(\frac{1}{1+e^{-x_i^T w_2}}\right)$$

$$= \sum_{i=1}^{n} \log(\sigma(x_i^T w_2))$$

Differentiating & equating to zero, we get

$$\nabla_w NLL_D(w_2) = -\sum \frac{1}{\sigma(x^T w_2)} . \sigma(x^T w_2)(1-\sigma(x^T w_2)).X = 0$$

$$\Rightarrow \boxed{\sigma(x^T w_2) = 1} \longrightarrow 3$$

Since $1 = I = I^T$ (identity matrix)

From 2 & 3

$$\sigma(w_1^T x) = \sigma(x^T w_2)$$

$\therefore \quad W_1^T x = x^T w_2$

$$\Rightarrow \boxed{\bar{w}_1 = w_2}$$

$\therefore$, Both approaches converge to same weights

2. Since logistic regression is the same as MLE on Bernoulli distributed data, for decision boundary,

$$p(y=1|x) = p(y=0|x) = 0.5$$

Now, $p(y|x) = h(x)^y (1-h(x))^{1-y}$

where $h(x) = \sigma(x^T w)$

$$\therefore p(y=1|x) = h(x) = 0.5$$

$$\Rightarrow \sigma(x^T w) = 0.5$$

$$\Rightarrow \frac{1}{1+e^{-x^T w}} = 0.5$$

$$\Rightarrow e^{-x^T w} = 1$$

$$\Rightarrow x^T w = 0$$

Similarly, for

$$P(y=0|x) = 1-h(x) = 0.5$$

$$\Rightarrow h(x) = 0.5$$

$$\Rightarrow \frac{1}{1+e^{-x^T w}} = 0.5$$

$$\Rightarrow e^{-x^T w} = 1$$

$$\Rightarrow x^T w = 0$$

Hence, the decision boundary is given by $x^T w = 0$

3. $p(y=1|x;\hat{w}) = \sigma(x^T\hat{w})$

Given all examples classified correctly,
$\Rightarrow x^T\hat{w} = 0$

$L(\hat{w}) = \prod p(y=1|x;\hat{w})$

$\therefore \log L(\hat{w}) = l(\hat{w}) = \sum_{i=1}^{n} \log(\sigma(x_i^T\hat{w}))$

$\Rightarrow l(c\hat{w}) = \sum_{i=1}^{n} \log(\sigma(x_i^T c\hat{w}))$

Given convergence

$\Rightarrow \nabla_w l(c\hat{w}) = \sum_{i=1}^{n} \frac{1}{\sigma(x_i^T c\hat{w})} \cdot \sigma(x_i^T c\hat{w}) \cdot (1-\sigma(x_i^T c\hat{w})) \cdot x_i^T = 0$

$\therefore \sigma(x_i^T c\hat{w}) = 1$

$\Rightarrow x_i^T c\hat{w} = 0$

$\Rightarrow c(x_i^T\hat{w}) = 0$

$\Rightarrow c(0) = 0$

$\Rightarrow c \in (-\infty, \infty)$

$\therefore$ Theres no single well defined optimal weight as c can scale infinitely

4. $$J_{logistic}(w) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-y_i w^T x_i}) + \lambda\|w\|^2$$

We know that

(i) e^z is convex $\forall z \in \mathbb{R}$

(ii) $\log(z)$ is convex $\forall z \in \mathbb{R}$

$\Rightarrow \log(1+e^z)$ is convex as $1+e^z \in \mathbb{R}$

(iii) w^2 is convex (quadratic)

Since sum of convex functions is also convex

$$J_{logistic}(w) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-y_i w^T x_i}) + \lambda\|w\|^2$$

is convex.

5. Please refer to the following functions

```python
def log_inv_sigmoid(z):
    c = min(-z)
    return c + np.logaddexp(np.zeros(z.shape) - c, -z - c)
```

```python
def f_objective(theta, X, y, l2_param=1):
    '''
    Args:
        theta: 1D numpy array of size num_features
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        l2_param: regularization parameter

    Returns:
        objective: scalar value of objective function
    '''
    margin = y * (X @ tl
    loss = (sum(log_inv,                                                    heta)
    return loss
```

```python
def log_inv_sigmoid(z):
    c = min(-z)
    return c + np.logaddexp(np.zeros(z.shape) - c, -z - c)
```

6. Please refer to the following image and attached jupyter file for function & training respectively

```python
def fit_logistic_reg(X, y, objective_function, l2_param=1):
    '''
    Args:
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        objective_function: function returning the value of the objective
        l2_param: regularization parameter

    Returns:
        optimal_theta: 1D numpy array of size num_features
    '''
    return minimize(objective_function, np.zeros(X.shape[1]), args=(X, y, l2_param)).x
```

7. Please refer to the following and jupyter notebook

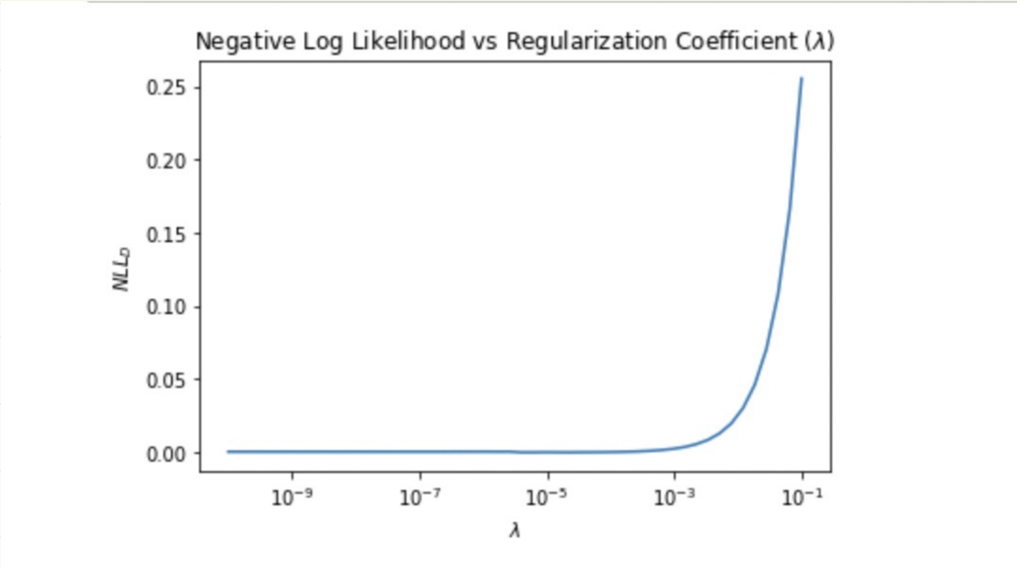
In [4]:
```python
# Find optimal regularization param on validation data
X_val = np.genfromtxt(path_to_data / "X_val.txt", delimiter=",")
X_val = standardize(X_val)
X_val = np.concatenate((np.ones((X_val.shape[0], 1)), X_val), axis=1)  # Add bias
y_val = np.genfromtxt(path_to_data / "y_val.txt", delimiter=",")

reg_candidates = np.logspace(-10, -1, num=50)
val_losses, thetas = [], []

for reg in reg_candidates:
    theta = fit_logistic_reg(X, y, f_objective, reg)
    thetas.append(theta)
    val_losses.append(negative_log_likelihood(X_val, theta))

plt.plot(reg_candidates, val_losses)
plt.xscale("log")
plt.xlabel(r"$\lambda$")
plt.ylabel("$NLL_{D}$")
plt.title("Negative Log Likelihood vs Regularization Coefficient ($\lambda$)")
plt.show()
```

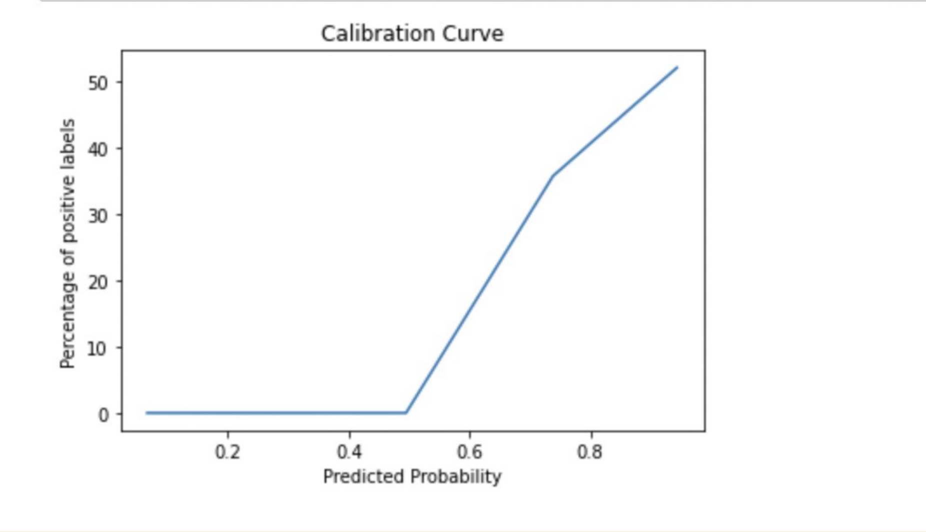
Training



NLL vs λ

Optimal l_2 param $= 4.714866 \times 10^{-7}$

8. As shaw belaw, in the calibration curve, the percent of positive labels stays 0, until we look at labels predicted with 0.5 probability or higher (i.e., $p(y=1|x) \geq 0.5$) We the observe a steady increase in the percentage of positive labels as we approach $p(y=1|x) = 1$



Please refer to the jupyter notebook snap for more details.

9. Given $p(z=H|\theta_1)=\theta_1$, $p(x=H|z=H,\theta_2)=\theta_2$

Since θ_1 & θ_2 are given parameters, it follows that

$p(z=H)=\theta_1$ & $p(x=H|z=H)=\theta_2$

Using Chain Rule

$\therefore p(x=H|\theta_1,\theta_2)=p(x=H)=p(z=H).p(x=H|z=H)$

$$\Rightarrow p(x=H|\theta_1,\theta_2)=\theta_1\theta_2$$

10. We write likelihood as

$$L_D=\prod_{i=1}^{N_n} p(x|\theta_1,\theta_2)$$

$$= p(x=H|\theta_1,\theta_2)^{n_h}(1-p(x=H|\theta_1,\theta_2))^{n_t}$$

$$=(\theta_1\theta_2)^{n_h}(1-\theta_1\theta_2)^{n_t}$$

11. For the derived log likelihood L_D, we cannot estimate θ_1, θ_2 using MLE as shown below

$$\nabla_{\theta_1\theta_2} L = \frac{d}{d\theta_1\theta_2}\left((\theta_1\theta_2)^{n_h}(1-\theta_1\theta_2)^{n_t}\right) = 0$$

$$\Rightarrow n_h(\theta_1\theta_2)^{n_h-1} - (n_h+n_t)(\theta_1\theta_2)^{n_h+n_t-1} = 0$$

$$\Rightarrow n_h(\theta_1\theta_2)^{n_h-1} = (n_h+n_t)(\theta_1\theta_2)^{n_h+n_t-1}$$

$$\Rightarrow \theta_1\theta_2^{n_t} = \frac{n_h}{n_h+n_t}$$

$$\boxed{\Rightarrow \theta_1\theta_2 = \log_{n_t}\frac{n_h}{n_h+n_t}}$$

Given that θ_1 & θ_2 can take infinitely values to satisfy the above condition, we cannot estimate their optimal individual values.

hw4_sol

March 27, 2022

1 Machine Learning : HW3

1.1 Q5-Q6

[1]:
```
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from pathlib import Path
from matplotlib import pyplot as plt

from hw4.logistic_code.logreg_skeleton import *
```

[2]:
```
# Load data
path_to_data = Path("") / "hw4" / "logistic_code"
X_train = np.genfromtxt(path_to_data / "X_train.txt", delimiter=",")
mean, std = np.mean(X_train, axis=0), np.std(X_train, axis=0)
y_train = np.genfromtxt(path_to_data / "y_train.txt", delimiter=",")
```

[3]:
```
def standardize(arr):
    return (arr - mean) / std

# Normalize
X = standardize(X_train)
X = np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)  # Add bias
y = y_train
```

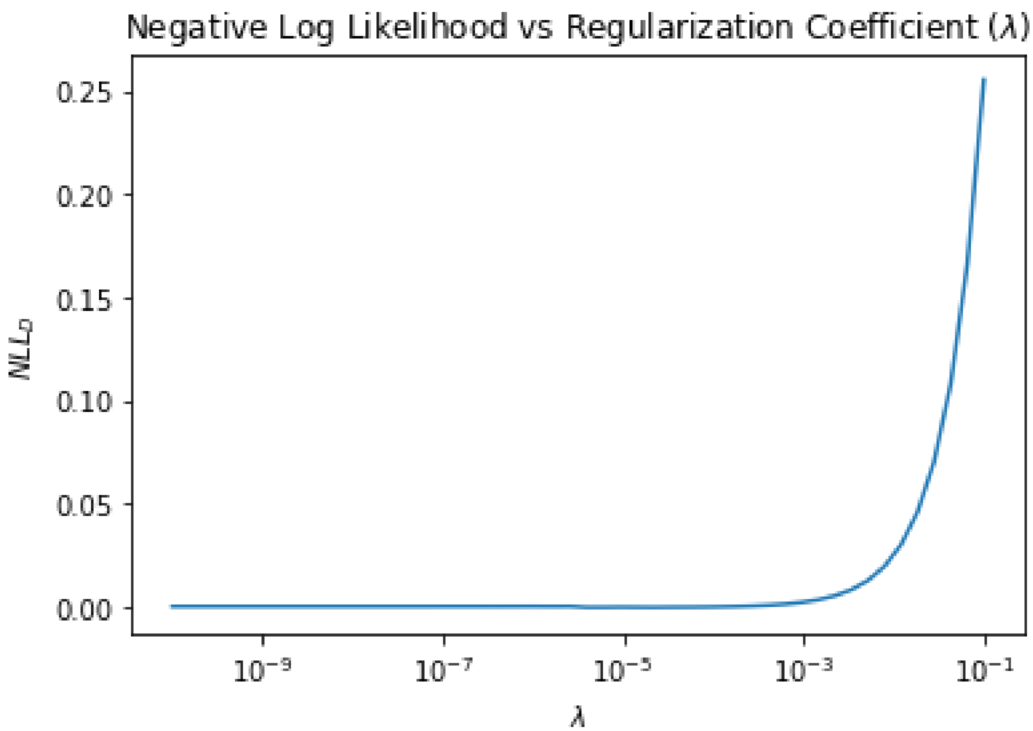
1.2 Q7

[7]:
```
# Find optimal regularization param on validation data
X_val = np.genfromtxt(path_to_data / "X_val.txt", delimiter=",")
X_val = standardize(X_val)
X_val = np.concatenate((np.ones((X_val.shape[0], 1)), X_val), axis=1)  # Add␣
 ↪bias
y_val = np.genfromtxt(path_to_data / "y_val.txt", delimiter=",")

reg_candidates = np.logspace(-10, -1, num=50)
val_losses, thetas = [], []
```

```
for reg in reg_candidates:
    theta = fit_logistic_reg(X, y, f_objective, reg)
    thetas.append(theta)
    val_losses.append(negative_log_likelihood(X_val, theta))

plt.plot(reg_candidates, val_losses)
plt.xscale("log")
plt.xlabel(r"$\lambda$")
plt.ylabel("$NLL_{D}$")
plt.title("Negative Log Likelihood vs Regularization Coefficient ($\lambda$)")
plt.savefig("nll_lambda.pdf")
```



[5]:
```
# Find l2 regularization param for optimal loss
print(reg_candidates[::-1][val_losses.index(min(val_losses[::-1]))])
```

```
4.7148663634573897e-07
```

1.3 Q8

[6]:
```
from sklearn.calibration import calibration_curve

optimal_theta = thetas[val_losses.index(min(val_losses))]
```

```
y_prob = np.exp(-log_inv_sigmoid(X_val @ optimal_theta))

fraction_of_positives, mean_predicted_value = calibration_curve(y_val, y_prob,␣
↪n_bins=5, normalize=True)
plt.plot(mean_predicted_value, [100* i for i in fraction_of_positives])
plt.title("Calibration Curve")
plt.xlabel("Predicted Probability")
plt.ylabel("Percentage of positive labels")
plt.show()
```