DSGA 1003_HW4_yp2201@nyu.edu

1. Show that the two approaches are equivalent, i.e. they will produce the same solution for w.

- ERM) $\hat{y} = \text{sign}(x^T w)$, $\ell_{\text{logistic}}(y, w) = \log(1 + \exp(-y w^T x))$

Empirical Risk: $\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \ell_{\text{logistic}}(y_i w^T x_i) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$

$$n \cdot \hat{R}_n(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

$$= \sum_{i=1}^{n} \log(1 + \exp(-y_i' w^T x_i))$$

$$= \begin{cases} \sum_{i=1}^{n} \log(1 + \exp(-w^T x_i)), & \text{if } y_i = 1 \\ \sum_{i=1}^{n} \log(1 + \exp(w^T x_i)), & \text{if } y_i = -1 \end{cases}$$

- MLE w Bernoulli) $p(y=1 \mid x; w) = 1/(1+\exp(-x^T w))$

Let's define $y_i' = \begin{cases} 1 & \text{if } y_i = 1 \\ 0 & \text{if } y_i = -1 \end{cases}$, for every x_i in dataset D,

D' : resulting collection of (x_i, y_i') pairs

$$NLL(w) = -\sum_{i=1}^{n} \{ y_i' \log(1/(1+\exp(-x_i^T w))) + (1 - y_i') \log(1 - 1/(1+\exp(-x_i^T w))) \}$$

$$= \sum_{i=1}^{n} \{ -y_i' \log(1/(1+\exp(-x_i^T w))) + (y_i' - 1) \log(1 - 1/(1+\exp(-x_i^T w))) \}$$

$$= \sum_{i=1}^{n} \{ y_i' \log(1+\exp(-x_i^T w)) + (1 - y_i') \log\left(\frac{1+\exp(-x_i^T w)}{\exp(-x_i^T w)} \right) \}$$

$$= \sum_{i=1}^{n} \{ y_i' \log(1+\exp(-x_i^T w)) + (1 - y_i') \log\left(1 + \frac{1}{\exp(-x_i^T w)} \right) \}$$

$$= \sum_{i=1}^{n} \{ y_i' \log(1+\exp(-x_i^T w)) + (1 - y_i') \log(1 + \exp(x_i^T w)) \}$$

$$= \begin{cases} \sum_{i=1}^{n} \log(1+\exp(-x_i^T w)), & \text{if } y_i = 1 \ (y_i' = 1) \\ \sum_{i=1}^{n} \log(1+\exp(x_i^T w)), & \text{if } y_i = -1 \ (y_i' = 0) \end{cases}$$

$\therefore n\hat{R}_n(w) = NLL(w)$ for all $w \in \mathbb{R}^d$,

Thus, two approaches are equivalent

Linearly Separable Data

In this problem, we will investigate the behavior of MLE for logistic regression when the data is linearly separable.

2. Show that the decision boundary of logistic regression is given by $\{x : x^T w = 0\}$. Note that the set will not change if we multiply the weights by some constant c.

Logistic regression predicts $\hat{y} = \text{sign}(x^T w)$

For all x, $\begin{cases} \hat{y} = 1, & \text{if } x^T w > 0 \\ \hat{y} = -1, & \text{if } x^T w < 0 \end{cases}$

Therefore, The decision boundary is given by

$$\{ x : x^T w = 0 \}$$

3. Suppose the data is linearly separable and by gradient descent/ascent we have reached a decision boundary defined by $\hat{w}$ where all examples are classified correctly. Show that we can always increase the likelihood of the data by multiplying a scalar c on $\hat{w}$, which means that MLE is not well-defined in this case. (Hint: You can show this by taking the derivative of $L(c\hat{w})$ with respect to c, where L is the likelihood function.)

Likelihood is given by $L(c\hat{w}) = \sum_{i=1}^{n} \log\left(1 + e^{-y_i (c\hat{w}_i)^T x_i}\right)$

$$\frac{\partial L}{\partial c} = \sum_{i=1}^{n} \frac{e^{-y_i (c\hat{w}_i)^T x_i} \times (-y_i \hat{w}_i^T x_i)}{1 + e^{-y_i (c\hat{w}_i)^T x_i}}$$

With $-y_i \hat{w}_i^T x_i \leq 0 \quad \forall i$, since $\hat{w}$ is separating the data

$\Rightarrow \frac{\partial L}{\partial c} \leq 0 \quad \forall c \Rightarrow L$ is a decreasing function of $c \in \mathbb{R}$

$\Rightarrow$ we can always make L bigger by choosing $c' < c$

($= L(w)$ decreases when c increases)

Regularized Logistic Regression

As we've shown in above, when the data is linearly separable, MLE for logistic regression may end up with weights with very large magnitudes. Such a function is prone to overfitting. In this part, we will apply regularization to fix the problem.

The ℓ_2 regularized logistic regression objective function can be defined as

$$\begin{aligned} J_{\text{logistic}}(w) &= \hat{R}_n(w) + \lambda \|w\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp\left(-y^{(i)} w^T x^{(i)}\right)\right) + \lambda\|w\|^2. \end{aligned}$$

2) 1)

4. Prove that the objective function $J_{\text{logistic}}(w)$ is convex. You may use any facts mentioned in the convex optimization notes.

1) $w \longrightarrow \lambda \|w\|^2$ is convex for $\lambda > 0$

2) $\nabla_w J = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i x_i e^{-y_i w^T x_i}}{1 + e^{-y_i w^T x_i}}$

Hessian) $\nabla_w^2 J = \frac{1}{n}\sum_{i=1}^{n} \left\{ \frac{-y_i^2 x_i x_i^T e^{-2y_i w^T x_i}}{(1 + e^{-y_i w^T x_i})^2} + \frac{y_i^2 x_i x_i^T e^{-y_i w^T x_i}}{(1 + e^{-y_i w^T x_i})} \right\}$

Lets set $\sigma(n) = \frac{1}{1+e^{-n}}$, as a sigmoid function $(0 \le \sigma(n) \le 1)$

$= \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T y_i^2 \sigma(-y_i w^T x_i)\left(1 - \sigma(-y_i w^T x_i)\right)$

- ✓ $x_i x_i^T$ is a positive semi-definite matrix
- ✓ $y_i^2 \sigma(-y_i w^T x_i)\left(1 - \sigma(-y_i w^T x_i)\right) \ge 0$
- ✓ Sum of positive semi-definite matrix is semi-positive definite

∴ the Hessian is positive semi-definite ⇒ convex

2 Coin Flipping with Partial Observability

Consider flipping a biased coin where $p(z = H \mid \theta_1) = \theta_1$. However, we cannot directly observe the result z. Instead, someone reports the result to us, which we denotey by x. Further, there is a chance that the result is reported incorrectly *if it's a head*. Specifically, we have $p(x = H \mid z = H, \theta_2) = \theta_2$ and $p(x = T \mid z = T) = 1$.

9. Show that $p(x = H \mid \theta_1, \theta_2) = \theta_1\theta_2$.

$$p(x=H \mid \theta_1,\theta_2) = p(x=H \mid z=H,\theta_2)\, p(z=H \mid \theta_2) + p(x=H \mid z=T,\theta_2)\, p(z=T \mid \theta_2)$$

$$(\, p(x=T \mid z=T) = 1 \Rightarrow p(x=H \mid z=T) = 0 \,)$$

$$= \theta_2 \cdot \theta_1 + 0$$

$$= \theta_1\theta_2$$

10. Given a set of reported results $\mathcal{D}_r$ of size N_r, where the number of heads is n_h and the number of tails is n_t, what is the likelihood of $\mathcal{D}_r$ as a function of θ_1 and θ_2.

$$p(\mathcal{D}_r \mid \theta_1,\theta_2) = \binom{n_h+n_t}{n_h} (\theta_1\theta_2)^{n_h} (1-\theta_1\theta_2)^{n_t}$$

Since $\begin{cases} p(x=H \mid \theta_1,\theta_2) = \theta_1\theta_2 \\ p(x=T \mid \theta_1,\theta_2) = 1-\theta_1\theta_2 \end{cases}$

11. Can we estimate θ_1 and θ_2 using MLE? Explain your judgment.

No, we can't estimate θ_1, θ_2 as θ_1 and θ_2 are not independently identifiable.
(We can only get the estimate of $\theta_1 \cdot \theta_2$)

Q5 ¶

Complete the f objective function in the skeleton code, which computes the objective function for Jlogistic(w). (Hint: you may get numerical overflow when computing the exponential literally, e.g. try e1000 in Numpy. Make sure to read about the log-sum-exp trick and use the numpy function logaddexp to get accurate calculations and to prevent overflow.

```
# numerical overflow check
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.exp(1000)
```

```
<ipython-input-1-f582a92b4683>:6: RuntimeWarning: overflow encountered in exp
  np.exp(1000)
```

```
inf
```

```
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
    '''

    # objective function = avg loss + reg term
    # avg_loss
    avg_loss = (1/X.shape[0]) * sum(np.logaddexp(0, -y*(X @ theta.T )))

    # reg term
    l2_reg = l2_param * (theta @ theta.T)

    # return scalar value of objective function
    objective = avg_loss + l2_reg

    return objective
```

Q6

Complete the fit logistic regression function in the skeleton code using the minimize function from scipy.optimize. Use this function to train a model on the provided data. Make sure to take the appropriate preprocessing steps, such as standardizing the data and adding a column for the bias term.

```
from scipy.optimize import minimize

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
    '''

    # initialize optimal theta
    optimal_theta = np.zeros(X.shape[1])

    # update optimal theta by using minimize function
    optimal_theta = minimize(objective_function, optimal_theta, args = (X, y, l2_param)).x

    return optimal_theta
```

```
# preprocessing steps: Source from HW2 code
def feature_normalization(train, test):

    # discard features that are constant in the training set
    remove = []
    for i in range(train.shape[1]):
        if len(set(train[:, i])) == 1:
            remove.append(i)
    for i in remove:
        np.delete(train, i, axis=1)
        np.delete(test, i, axis=1)

    # min-max scaling
    train_normalized = np.array(train.shape)
    test_normalized = np.array(test.shape)
    train_max_vals = np.max(train, axis=0)
    train_min_vals = np.min(train, axis=0)

    # use train min-max to transform both train/test dataset
    train_normalized = (train - train_min_vals) / (train_max_vals - train_min_vals)
    test_normalized = (test - train_min_vals) / (train_max_vals - train_min_vals)

    return train_normalized, test_normalized
```

```
# import txt files
X_train = np.loadtxt('./X_train.txt', delimiter = ',')
y_train = np.loadtxt('./y_train.txt', delimiter = ',')
X_val = np.loadtxt('./X_val.txt', delimiter = ',')
y_val = np.loadtxt('./y_val.txt', delimiter = ',')
```

```
# standardizing the data
X_train_normalized, X_val_normalized = feature_normalization(X_train, X_val)
```

```
# adding a column for the bias term
X_train_normalized = np.hstack((X_train_normalized, np.ones((X_train_normalized.shape[0], 1))))
X_val_normalized = np.hstack((X_val_normalized, np.ones((X_val_normalized.shape[0], 1))))
```

```
# as we assume outcome space Y± = {-1, 1}, convert y = 0 values into -1
y_train[y_train == 0] = -1
y_val[y_val == 0] = -1
```

```
# Train model
# we need to define f_objective
optimal_theta = fit_logistic_reg(X_train_normalized, y_train, f_objective, l2_param=1)
optimal_theta
```

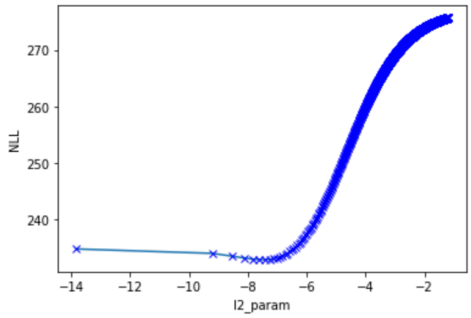
```
array([ 0.00098731,  0.00086963,  0.00030947,  0.02207394,  0.00024183,
        0.00074032,  0.00011989,  0.00085942,  0.00090577, -0.01224866,
        0.0019563 ,  0.000236  ,  0.00459849,  0.00012707,  0.00047895,
        0.00117241,  0.00037255, -0.00044841, -0.00044868, -0.00069619,
        0.00178106])
```

Q7

Find the l2 regularization parameter that minimizes the log-likelihood on the validation set.
Plot the log-likelihood for different values of the regularization parameter.

```
import matplotlib.pyplot as plt

def neg_log_likelihood(theta, X, y, l2_param):

    # From Q1, we know that n*Rn(w) = NLL(w), in terms of w
    # avg_loss
    avg_loss = (1/X.shape[0]) * sum(np.logaddexp(0, -y*(X @ theta.T )))

    # return scalar value of objective function
    objective = avg_loss

    # multiplying by n and return
    return X.shape[0] * objective

# creating nll empty set, and l2 param range
NLL_res = []
l2_param_range = np.arange(0.000001, 0.3, 0.0001)

# for given l2 param range, calculate theta
for l2_param in l2_param_range:
    optimal_theta = fit_logistic_reg(X_train_normalized, y_train, f_objective, l2_param)
    NLL_res.append(neg_log_likelihood(optimal_theta, X_val_normalized, y_val, l2_param))

# plotting the answers
print('Best l2 param:{}, NLL: {}'.format(l2_param_range[np.argmin(NLL_res)], np.min(NLL_res)))
plt.plot(np.log(l2_param_range), NLL_res)
plt.plot(np.log(l2_param_range), NLL_res, 'bx')
plt.xlabel('l2_param')
plt.ylabel('NLL')
plt.show()
```

```
Best l2 param:0.000501, NLL: 232.9126744529017
```



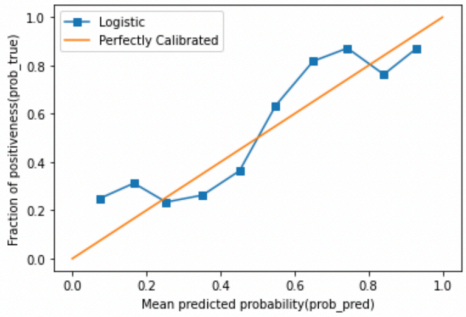
Q8

[Optional]It seems reasonable to interpret the predictionf(x)=ϕ(wTx)=1/(1+e−wTx) as the probability that y = 1, for a randomly drawn pair (x, y). Since we only have a finite sample (and we are regularizing, which will bias things a bit) there is a question of how well “calibrated” our predicted probabilities are. Roughly speaking, we say f(x) is well calibrated if we look at all examples (x,y) for which f(x) ≈ 0.7 and we find that close to 70% of those examples have y = 1, as predicted... and then we repeat that for all predicted probabilities in (0, 1). To see how well-calibrated our predicted probabilities are, break the predictions on the validation set into groups based on the predicted probability (you can play with the size of the groups to get a result you think is informative). For each group, examine the percentage of positive labels. You can make a table or graph. Summarize the results. You may get some ideas and references from scikit-learn’s discussion.

Ans) LogisticRegression returns well calibrated predictions by default as it directly optimizes Log loss

```
# Best l2 param: 0.000501
from sklearn.calibration import calibration_curve

optimal_theta = fit_logistic_reg(X_train_normalized, y_train, f_objective, l2_param = 0.000501)
y_pred = 1/(1+np.exp(-(X_val_normalized @ optimal_theta.T)))

prob_true, prob_pred = calibration_curve(y_val, y_pred, normalize=False, n_bins=10)
```

```
plt.plot(prob_pred, prob_true, marker = 's', label = 'Logistic')
plt.plot([0, 1], [0, 1], label = 'Perfectly Calibrated')
plt.xlabel('Mean predicted probability(prob_pred)')
plt.ylabel('Fraction of positiveness(prob_true)')
plt.legend()
plt.show()
```



The percentage of positive labels for each group

```
# The percentage of positive labels for each group
for idx, value in enumerate(prob_true):
    print('Group {}) Percentage of positive labels: {:.2f}'.format(idx+1, value))
```

```
Group 1) Percentage of positive labels: 0.25
Group 2) Percentage of positive labels: 0.31
Group 3) Percentage of positive labels: 0.23
Group 4) Percentage of positive labels: 0.26
Group 5) Percentage of positive labels: 0.36
Group 6) Percentage of positive labels: 0.63
Group 7) Percentage of positive labels: 0.82
Group 8) Percentage of positive labels: 0.87
Group 9) Percentage of positive labels: 0.76
Group 10) Percentage of positive labels: 0.87
```