HW 4 Probabilistic Models

Joby George (jg6615)

Due 3/23/2022

Problem 1 Prompt:

Prove equivalence of ERM for binary classification using logistic loss and probabilistic approach to logistic regression results in the same minimization problem

Problem 1 Answer

MLE Expression for logistic Regression

Using slides 45 and 46 from week two of lecture, we know the logistic log function is

$$L(\theta) = \frac{1}{n}\sum_{i}^{n} log(1 + e^{-m}) \; and$$

where n = the number of data points

$$m = y_i \theta^T x_i$$

Which is equivalent to:

$$L(\theta) = \frac{1}{n}\sum_{i}^{n} log(1 + e^{-y_i \theta^T x_i}) \; \mathrm{where} \; y \; \in -1, 1$$

Which is the expression we would like to minimize to minimize empirical risk.

Negative Log-likelihood for logistic regression

Case 1 Y_i = 0, (-1 in MLE Outcome Space)

From slide 19 we observe the log-likelihood we would like to maximize:

$$NLL = -\sum_{i}^{n} y_i log(f(\theta^T x_i)) + (1 - y_i)(log(1 - f(\theta^T x_i)) \; where \; y_i \; \in 0, 1 \; and$$

$$f(\eta) = \frac{1}{1+e^{\eta}}$$

if y_i = 0, we observe this to be:

$$NLL = \sum_{i}^{n} -log(1 - f(\theta^T x_i))$$

plugging in our logistic link function, if y_i = 0 we get:

$$NLL = \sum_{i}^{n} -log(1 - \frac{1}{1+e^{(-\theta^T x_i)}})$$

setting a common denominator and expanding our logistic, we get:

$$NLL = \sum_{i}^{n} -log(\frac{1+e^{(-\theta^T x_i)} - 1}{1+e^{(-\theta^T x_i)}})$$

simplifying this we get

$$NLL = \sum_{i}^{n} -log(\frac{e^{(-\theta^T x_i)}}{1+e^{(-\theta^T x_i)}})$$

multiplying the numerator and denominator by one, expressed as:

$$\frac{e^{(\theta^T x_i)}}{e^{(\theta^T x_i)}}$$

we get:

$$NLL = \sum_{i}^{n} -log(\frac{e^{0}}{(1+e^{(-\theta^T x_i)})(e^{(\theta^T x_i)})})$$

which simplifies to

$$NLL = \sum_{i}^{n} -log(\frac{1}{1+e^{(\theta^T x_i)}})$$

if we apply the negative sign to the log by taking the reciprocal of it's input, and scale it by a factor of $\frac{1}{n}$ we end with our final expression:

$$NLL = \frac{1}{n}\sum_{i}^{n} log(1+e^{\theta^T x_i})$$

This is equivalent to the MLE when y_i = -1:

$$L(\theta) = \frac{1}{n}\sum_{i}^{n} log(1 + e^{\theta^T x_i})$$

Case 1 Y_i = 1, (1 in MLE Outcome Space)

$$NLL = \sum_{i}^{n} -log(f(\theta^T x_i))$$

plugging in our logistic link function, if y_i = 1 we get:

$$NLL = \sum_{i}^{n} -log(\frac{1}{1 + e^{(-\theta^T x_i)}})$$

if we apply the negative sign to the log by taking the reciprocal of it's input, and scale it by a factor of $\frac{1}{n}$ we end with our final expression:

$$NLL = \frac{1}{n}\sum_{i}^{n} log(1 + e^{(-\theta^T x_i)})$$

This is equivalent to the MLE when y_i = 1:

$$L(\theta) = \frac{1}{n}\sum_{i}^{n} log(1 + e^{-\theta^T x_i})$$

Q.E.D

Problem 2 Prompt:

Show that the decision boundary of logistic regression is given by x : x^Tw = 0. Note that the set will not change if we multiply the weights by some constant c.

Problem 2 answer

First let's prove that the decision boundary equals x^Tw, we start with the log odds ratio from the slides:

$$log(\frac{P(Y = 1|X)}{P(Y = 0|X)})$$

plugging in our formulas for those two expressions, we get:

$$log(\frac{\frac{1}{1+e^{(-x^Tw)}}}{1-\frac{1}{1+e^{(-x^Tw)}}})$$

Focusing only on the denominator P(Y=0|X), we can simplify it by setting a common denominator and algebraically manipulating it.

$$denominator = \frac{1+e^{(-x^Tw)}-1}{1+e^{(-x^Tw)}}$$

$$denominator = \frac{e^{(-x^Tw)}}{1+e^{(-x^Tw)}}$$

giving us

$$log(\frac{\frac{1}{1+e^{(-x^Tw)}}}{\frac{e^{(-x^Tw)}}{1+e^{(-x^Tw)}}})$$

simplifying this by applying the division we get

$$log(\frac{1}{e^{(-x^Tw)}})$$

multiplying the numerator and denominator by 1 in the form of

$$\frac{e^{(x^Tw)})}{e^{(x^Tw)}}$$

we get

$$log(\frac{e^{(x^Tw)}}{1}) = x^Tw$$

Thus proving our log odds function is the decision boundary.

Next, to prove that our decision boundary is 0. By definition, the decision boundary is where the P(y=1|x) = P(y=0|x) = .5

Plugging this into our equation for P(Y=1|X), we observe:

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

$$2 = 1+e^{(-x^Tw)}$$

$$1 = e^{(-x^Tw)}$$

$$log(1) = -x^T w$$

$$x^T w = 0$$

Solving for P(Y=0|X), we observe:

$$1 - \frac{1}{1 + e^{(-x^T w)}} = \frac{1}{2}$$

simplifying via a common denominator, we get:

$$\frac{e^{(-x^T w)}}{1 + e^{(-x^T w)}} = \frac{1}{2}$$

Cross multiplying we get:

$$2e^{(-x^T w)} = 1 + e^{(-x^T w)}$$

subtracting one common term we get

$$e^{(-x^T w)} = 1$$

taking the log of both sides we get:

$$x^T w = 0$$

This shows that the decision boundary, defined by when the log odds of both classes are equivalent, must be when $x^T w$ = 0

Furthermore we can scale our weights by some constant c to get an equivalent but differently shaped decision boundary, meaning many equally accurate decision boundaries exist for linearly separable data.

Q.E.D

Problem 3 Prompt:

Assume the data are linearly separable and we have reached a $\hat{w}$ that perfectly classifies the data. 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.)

Problem 3 Answer

$$L(c\hat{w}) = \sum_{i}^{n} y_i log(f(c\hat{w}^T x_i)) + (1 - y_i)(log(1 - f(c\hat{w}^T x_i)) \; where \; y_i \; \in 0, 1 \; and$$

$$L(c\hat{w}) \sum_{i}^{n} y_i log(\frac{1}{1 + e^{(-c\hat{w}^T x)}}) + (1 - y_i)(log(1 - \frac{1}{1 + e^{(-c\hat{w}^T x)}}) \; where \; y_i \; \in 0, 1 \; and$$

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

Simplifying this we get:

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

Since all of our predictions are correct, we note that our likelihood function can always increase.

When y_i = 1 we see:

$$\frac{\delta L(c\hat{w})}{\delta c} = \frac{\hat{w}^T x_i}{1 + e^{c\hat{w}^T x_i}}$$

Since we are correct in our prediction and y=1, the $\hat{w}^T x_i$ must be positive, meaning we can always increase the likelihood by manipulating c to be a smaller denominator. Furthermore we know the denominator is always positive as well as 1 + e^x is lower bounded by 1.

When y_i = 0 we see:

$$\frac{\delta L(c\hat{w})}{\delta c} = \frac{-\hat{w}^T x_i}{1 + e^{c\hat{w}^T x_i}}$$

Since y_i = 0, $\hat{w}^T x_i$ must be negative, and when we multiply it by -1, we get a positive number in our numerator. Again, this means we can manipulate c to increase the likelihood by making the denominator smaller.

This means that the MLE is not well defined.

Q.E.D.

Problem 4 Prompt:

Prove the objective function $J_{logistic}$ is convex.

$$J_{logistic} = \hat{R}_n(w) + \lambda||w||^2$$

Problem 4 Answer

From Professor Rosenberg's notes on convex optimization page 5 we see that the Log-Sum-Exp (LSE) is convex

$$LSE = log(e^{x_1} + .. e^{x_n})$$

Since $\hat{R}_n(w)$ = $\sum_i^n$(log(1+$e^{-y_i w^T x_i}$), which can be re-expressed as $\sum_i^n$(log(e^0+$e^{-y_i w^T x_i}$), meaning that $\hat{R}_n$(w) is convex

Additionally, on the same page of the notes linked above we observe that norms are convex as well.

The sum of two convex functions must be convex proving that our objective function $J_{logistic}$ is convex

Q.E.D.

Problem 5 Prompt

Complete the f objective function in the skeleton code, which computes the objective function for $J_{logistic(w)}$. (Hint: you may get numerical overflow when computing the exponential literally, e.g. try e^{1000} 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.

Problem 5 answer

Done below

In [2]:

```
import numpy as np
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
    '''
    n = X.shape[0]
    scalar = 1/n
    loss = 0
    for i in range(n):
        """
        logaddexp adds the log(e^(input1), e^input2)
        since we want it to be log(1+e^(-y*theta^Tx)) we need to actually pas
        0 in as our first input so e^(0) = 1
        """
        loss+=np.logaddexp(0,-y[i]*np.dot(theta,X[i]))
    loss*=scalar
    return(loss+l2_param*np.dot(theta,theta))
```

Problem 6 Prompt

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.

Problem 6 Answer

```
function completed in the next cell
data loading and preprocessing completed in the cell after
training the model done in the third cell down
```

In [7]:

```
### Finishing the function
from scipy.optimize import minimize
from functools import partial
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
    '''
    """logic
            create an initial feature vector of 1's
            create a partial objective function using functools.partial that
            X data, Y data and l2 parameter
            call scipy.optimize.minimize on the objective function passing in
            feature vector
            return the optimized weights
    """

    num_feats = X.shape[1]
    w_initial = np.ones(num_feats)
    obj_func = partial(objective_function, X=X, y=y, l2_param=l2_param)
    """
    minimize function takes in the function to minimize, with an initial weig
    the optimal feature values are obtained with the .x attribute
    """
    w_star = minimize(obj_func,w_initial).x
    return(w_star)
```

In [8]:

```
### loading and cleaning data
path = 'logistic-code/'
file_names = ['X_train.txt', 'X_val.txt', 'y_train.txt', 'y_val.txt']
path_list = [path + file_name for file_name in file_names]

for i in range(len(path_list)):
    if i == 0:
        with open(path_list[i]) as textFile:
            x_train = [line.split() for line in textFile]
        #change data type to reflect an array as data are comma seperated
        x_train = [[float(value) for value in row[0].split(',')] for row in x
    elif i == 1:
        with open(path_list[i]) as textFile:
            x_val = [line.split() for line in textFile]
        x_val = [[float(value) for value in row[0].split(',')] for row in x_v
    elif i == 2:
        with open(path_list[i]) as textFile:
                y_train = [line.split() for line in textFile]
        y_train = [[float(value) for value in row[0].split(',')] for row in y
    else:
        with open(path_list[i]) as textFile:
            y_val = [line.split() for line in textFile]
        y_val = [[float(value) for value in row[0].split(',')] for row in y_v



#convert data into numpy arrays
x_train, x_val, y_train, y_val,   = np.array(x_train), np.array(x_val), np.ar

#preprocessing the data: standardizing the input data
from sklearn.preprocessing import StandardScaler

SS = StandardScaler()
x_train = SS.fit_transform(x_train)
x_val = SS.transform(x_val)

##preprocessing the data: adding a bias column to x_train and x_val using np.
x_train = np.hstack((x_train, np.ones((x_train.shape[0],1))))
x_val = np.hstack((x_val, np.ones((x_val.shape[0],1))))

##change y train and val labels to -1 from 0
y_train[y_train == 0 ] = -1
y_val[y_val==0] = -1
```

In [9]:

```
theta = fit_logistic_reg(x_train, y_train, f_objective, l2_param =1)
```

Out[9]:

```
array([ 0.00095627, -0.00029963,  0.00302671,  0.10532759, -0.00358739,
       -0.00135854, -0.00385292, -0.00079012, -0.00114412, -0.07178437,
        0.00654884, -0.00451104,  0.01124929, -0.0038644 , -0.00271266,
        0.00150356, -0.00278402, -0.00919061, -0.00682285, -0.01027484,
        0.00281864])
```

Problem 7 Prompt

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.

Problem 7 Answer

done below

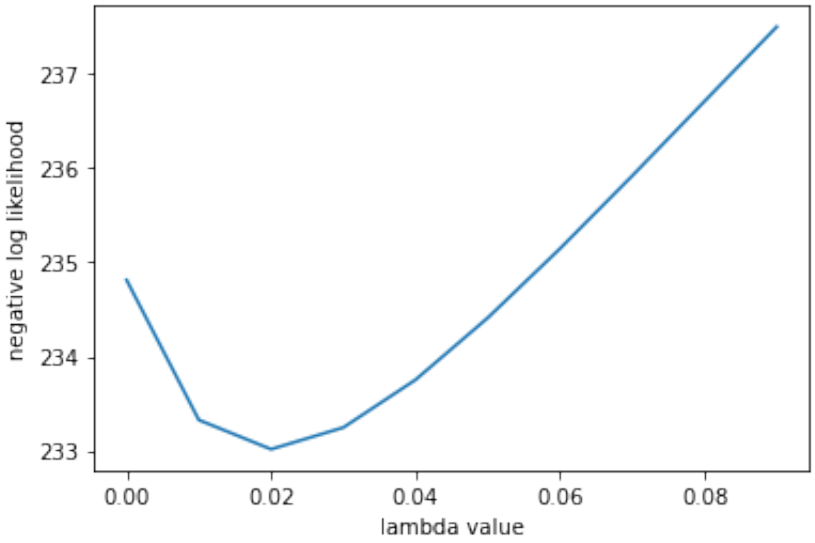
In [10]:

```
def negative_log_likelihood(theta,X,y):
    n = X.shape[0]
    loss = 0
    for i in range(n):
        loss +=np.logaddexp(0,-y[i]*np.dot(theta,X[i]))
    return(loss)

##initially started with a linspace between 0 and 2
#and noticed decreasing validation performance, so did a narrower range betwe

#lambda_list = np.linspace(0,2,10)
lambda_list = np.arange(0,.1,.01)


result_dict = dict()
for lam in lambda_list:
    theta = fit_logistic_reg(x_train, y_train,f_objective, l2_param=lam)
    result_dict[lam] = negative_log_likelihood(theta,x_val,y_val)
```

In [17]:

```
import matplotlib.pyplot as plt
x_axis = list(result_dict.keys())
y_values = list(result_dict.values())
plt.plot(x_axis,y_values)
plt.xlabel('lambda value')
plt.ylabel('negative log likelihood')
```

Out[17]: Text(0, 0.5, 'negative log likelihood')



In [12]:
```
min(result_dict, key=result_dict.get)
#optimal lambda = .02
```

Out[12]: 0.02

Problem 8 Prompt

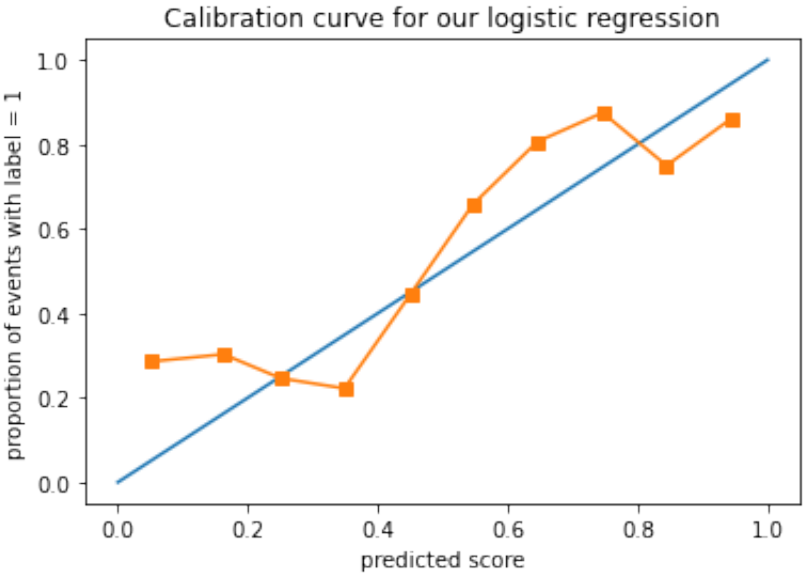
Plot the calibration of your model's probabilistic output. Calibration refers to the proportion of Y = 0 events occur for a given model score (i.e. if the score is .3, we would expect 30% of events to have a label of 0)

Problem 8 answer

done below

In [15]:

```
from sklearn.calibration import calibration_curve

#plot a calibration line that shows y = x from 0 to 1
plt.plot([0,1],[0,1])
optimal_theta = fit_logistic_reg(x_train, y_train, f_objective, l2_param = .0
y_pred = np.dot(x_val, optimal_theta)
for i in range(x_val.shape[0]):
    y_pred[i] = 1/(1+np.exp(-y_pred[i]))
    #this is not a 0 - 1 range, lets scale to 0 and 1
y_pred = (y_pred - y_pred.min())/(y_pred.max()-y_pred.min())
pos_fraction, mean_pred_val = calibration_curve(y_val, y_pred, n_bins = 10)
plt.plot(mean_pred_val, pos_fraction, 's-', label ='accuracy for predicted sc
plt.xlabel('predicted score')
plt.ylabel('proportion of events with label = 1')
plt.title('Calibration curve for our logistic regression')
plt.show()
```



Problem 9

Consider flipping a biased coin where p (z= H| θ_1) = θ_1. However, we can't observe the flip directly, and have it reported to us. There is a probability the report is incorrect **if the outcome is a head**, denoted by x: p(x = H|z=H,θ_2) = θ_2. If the outcome is a tails, it will always be correctly reported.

Show that P(x=h|θ_1.θ_2) = θ_1*θ_2

The probability of the report being heads, given the probability of the coin is heads (θ_1) and the probability of the report being accurate is (θ_2)

Problem 9 answer

From this situation, we know that the probability of incorrectly reporting heads is conditionally independent given z = H. i.e. P(x=H|z=H, θ_2) = P(x=H|z=H, θ_2, θ_1) = θ_2

Furthermore, we know that the probability of getting heads is independent from the probability of reporting heads. P(z=H|θ_1) = P(z=H|θ_1, θ_2) = θ_1

Expressing this, we observe:

$$\theta_1\theta_2 = P(z = H|\theta_1) * P(x = H|z = H, \theta_2)$$

Applying conditional independence to our first and second terms we get:

$$\theta_1\theta_2 = P(z = H|\theta_1, \theta_2) * P(x = H|z = H, \theta_2, \theta_1)$$

By Bayes Rule this equals

$$\theta_1\theta_2 = P(z = H|\theta_1, \theta_2) * \frac{P(x = H \cap z = H \cap \theta_2 \cap \theta_1)}{P(Z = H \cap \theta_2 \cap \theta_1)}$$

By chain rule we get:

$$\theta_1\theta_2 = P(z = H|\theta_1, \theta_2) * \frac{P(\theta_1) * P(\theta_2|\theta_1) * P(x = H|\theta_2, \theta_1)}{P(\theta_1)P(\theta_2|\theta_1)P(Z = H|\theta_1, \theta_2)}$$

We observe that two terms in the numerator and denominator cancel each other out, and the last term in the denominator is canceled by the multiplication, giving us.

$$\theta_1\theta_2 = P(x = H|\theta_1, \theta_2)$$

Q.E.D

Problem 10 Prompt

Given a set of reported results D_r of size N, where r = the number of reported heads = h and the number of reported tails = t, what is the likelihood of D_r as a function of θ_1 and θ_2

Problem 10 Answer

$$L(D_r) = P(x = H|\theta_1, \theta_2)^r * P(x = T|\theta_1, \theta_2)^t$$

$$P(x = H|\theta_1, \theta_2)^r = (\theta_1\theta_2)^r$$

$$P(x = T|\theta_1, \theta_2)^r = (1 - \theta_1\theta_2)^t$$

Therefore

$$L(D_r) = (\theta_1\theta_2)^r(1 - \theta_1\theta_2)^t$$

Q.E.D.

Problem 11 Prompt

Can we estimate θ_1 and θ_2 using MLE, explain your judgment

Problem 11 answer

We cannot adequately estimate these two parameters with MLE given our the way our data are reported back to us.

For example, let's say θ_1 = 1 and θ_2 = 0, our report would be entirely tails.

There are many solutions to this using an MLE perspective that are radically different, θ_1 = 0 and θ_2 can take any value (as we never observe a heads flip with this coin, so there was no distortion) or the true parameter distribution of θ_1 = 1 and θ_2 = 0.

Even in less extreme examples with a balance of heads and tails, we can split weight between our two parameters to end up with the same expected report, giving us unreliable estimators. Using a Bayesian approach with different priors and updating the prior throughout the sequence of events would be a more precise way to estimate the two parameters.

Showing this mathematically:

$$Log(L(D_r) = rlog(\theta_1\theta_2) + tlog(1 - \theta_1\theta_2)$$

Differentiating with respect to θ_1 and setting to zero

$$0 = \frac{r}{\theta_1} - \frac{t\theta_2}{1 - \theta_1\theta_2}$$

Solving for θ_1

$$\frac{t\theta_2}{1 - \theta_1\theta_2} = \frac{r}{\theta_1}$$

$$t\theta_2\theta_1 = r - r\theta_1\theta_2$$

$$t\theta_2\theta_1 + r\theta_1\theta_2 = r$$

$$\theta_2\theta_1(t + r) = r$$

$$\theta_2\theta_1 = \frac{r}{t + r}$$

$$\theta_1 = \frac{r}{t + r} * \frac{1}{\theta_2}$$

This means that the optimal θ_1 is dependent on θ_2 which we will also observe is dependent on the optimal θ_1

Differentiating with respect to θ_2 and setting to zero:

$$0 = \frac{r}{\theta_2} - \frac{t\theta_1}{1 - \theta_1\theta_2}$$

Solving for θ_2

$$\frac{t\theta_1}{1 - \theta_1\theta_2} = \frac{r}{\theta_2}$$

$$t\theta_1\theta_2 = r - r\theta_1\theta_2$$

$$t\theta_1\theta_2 + r\theta_1\theta_2 = r$$

$$\theta_1\theta_2(t + r) = r$$

$$\theta_1\theta_2 = \frac{r}{t + r}$$

$$\theta_2 = \frac{r}{t + r} * \frac{1}{\theta_1}$$

This shows MLE is not a good estimator of these two parameters as we would need to know one in the first place for our MLE estimate for the other to be reasonable.

Q.E.D

In [ ]: