Homework 4: Probabilistic models

Due: Friday, March 25, 2022 at 11:59PM EST

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g.LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better.

---

1 Logistic Regression

Consider a binary classification setting with input space $\mathcal{X} = \mathbb{R}^d$, outcome space $\mathcal{Y}_{\pm} = \{-1, 1\}$, and a dataset $\mathcal{D} = \big((x^{(1)}, y^{(1)}), \mathcal{D}ots, (x^{(n)}, y^{(n)})\big)$.

Equivalence of ERM and probabilistic approaches

In the lecture we derived logistic regression using the Bernoulli response distribution. In this problem you will show that it is equivalent to ERM with logistic loss.

ERM with logistic loss.

Consider a linear scoring function in the space $\mathcal{F}_{\text{score}} = \big\{x \mapsto x^T w \mid w \in \mathbb{R}^d\big\}$. A simple way to make predictions (similar to what we've seen with the perceptron algorithm) is to predict $\hat{y} = 1$ if $x^T w > 0$, or $\hat{y} = \text{sign}(x^T w)$. Accordingly, we consider margin-based loss functions that relate the loss with the margin, $yx^T w$. A positive margin means that $x^T w$ has the same sign as y, i.e. a correct prediction. Specifically, let's consider the **logistic loss** function $\ell_{\text{logistic}}(y, w) = \log\big(1 + \exp(-yw^T x)\big)$. This is a margin-based loss function that you have now encountered several times. Given the logistic loss, we can now minimize the empirical risk on our dataset $\mathcal{D}$ to obtain an estimate of the parameters, $\hat{w}$.

MLE with a Bernoulli response distribution and the logistic link function.

As discussed in the lecture, given that $p(y = 1 \mid x; w) = 1/(1+\exp(-x^T w))$, we can estimate w by maximizing the likelihood, or equivalently, minimizing the negative log-likelihood ($\text{NLL}_{\mathcal{D}}(\text{w})$ in short) of the data.

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

   We know from Lecture 2 and the problem statement, we know that the definition of the logistic log function is as follows:

   $$\ell_{\text{logistic}}(y, w) = \log\big(1 + \exp(-yw^T x)\big) \;\; thus \;\; L(w) = \frac{1}{n}\sum_{i=1}^{n} log\,\big(1 + \exp(-y_i w^T x_i)\big)$$

   Where $L(w)$ is the empirical risk minimizer and $y \in \{-1, 1\}$

   For the MLE approach we must calculate the product of the probabilities of seeing our data-points: $P(y_i|x; w)$. We can simplify by taking the log of our MLE and making it negative (for purposes of minimization, rather than maximization) thus making our problem minimizing the NLL. We know that $p(y = 1 \mid x; w) = 1/(1 + \exp(-x^T w))$ and

thus $p(y = 0 \mid x; w) = 1 - \frac{1}{(1+\exp(-x^T w)}$ where $y \in \{0, 1\}$ Therefore, we can derive our MLE expression by doing the following:

$$\begin{aligned} NLL(w) &= -MLE(w) \\ &= -\prod_{i=1}^{n} P(y|x_i; w) \\ &= -\sum_{i=1}^{n} log(P(y|x_i; w)) \\ &= -\sum_{i=1}^{n} log\ y_i \left(\frac{1}{1+e^{-x_i^T w}}\right) + (1-y_i) log\left(1 - \frac{1}{1+e^{-x_i^T w}}\right) \end{aligned} \tag{1}$$

Considering a single observation, (x_i, y_i) we can evaluate the above expression for our two cases: $y_i = 1$ and $y_i = 0$.

When $y = 0$, the NLL expression simplifies to:

$$-log(1-\frac{1}{1+e^{-x_i^T w}}) = -log(\frac{e^{-x_i^T w}}{1+e^{-x_i^T w}}) = -log(\frac{e^{-x_i^T w}}{1+e^{-x_i^T w}} \times \frac{e^{x_i^T w}}{e^{x_i^T w}}) = -log(\frac{1}{1+e^{x_i^T w}}) = log(1+e^{x_i^T w})$$

Which is the same value as our ERM approach: $\ell_{\text{logistic}}(y, w) = \log\left(1 + \exp(-yw^T x)\right)$ (whereas in MLE a negative label is $y = 0$ then in ERM $y = -1$)

Likewise in the other case of a positive label, so in NLL terms $y = 1$, and in ERM terms, $y = 1$.

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

Which again, is the same result that you get when using the ERM approach:

$$ERM(w) = log(1+e^{-yx^T w})$$

Thus we have shown that in our two cases, positive or negative labels, both ERM and NLL results in the same function.

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 $\{x \colon x^T w = 0\}$. Note that the set will not change if we multiply the weights by some constant c.

   We know that the decision boundary of logistic regression is defined as:

   $$P(y = 1|x; w) = P(y = 0|x; w) = \frac{1}{2}$$

   Using the definition for a positive label: $P(y = 1|x; w) = 1/(1 + e^{-w^T x})$ we can show why

the decision boundary can be defined as $x^T w = 0$:

$$\begin{aligned} \frac{1}{1+e^{-w^T x}} &= \frac{1}{2} \\ 2 &= 1 + e^{-w^T x} \\ log(1) &= log(e^{-w^T x}) \\ 0 &= -w^T x \\ x^T w &= 0 \end{aligned} \tag{2}$$

Furthermore, we can reach the same conclusion using the log odds definition:

$$\begin{aligned} log\left(\frac{P(y=1|x;w)}{P(y=1|x;w)}\right) &= w^T x \\ log\left(\frac{1/2}{1-1/2}\right) &= w^T x \\ log(1) = 0 &= w^T x \end{aligned} \tag{3}$$

Since our linear boundary is expressed as $x^T w = 0$, we can modify our expression and still get a linearly seperating decision boundary. Considering the scalar $\alpha \in \mathbb{R}$ then:

$$x^T \alpha w = 0 \rightarrow \langle x, \alpha w \rangle = \alpha \langle x, w \rangle = \alpha \times 0 = 0$$

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.)

   We can show this property by multiplying our score $x^T w$ with a constant $c \in \mathbb{R}$. Taking the derivative with respect to c:

$$\begin{aligned} L(cw) &= \sum_{i=1}^{n} P(y_i|x_i;cw) \\ L(cw) &= -\sum_{i=1}^{n} log\ y_i \left(\frac{1}{1+e^{-x_i^T w}}\right) + (1-y_i) log\left(1 - \frac{1}{1+e^{-x_i^T w}}\right) \\ \frac{\partial L(cw)}{\partial c} &= -\sum_{i=1}^{n} y_i \left(\frac{w^T x_i}{1+e^{-x_i^T cw}}\right) + (1-y_i) log\left(\frac{-w^T x_i \times e^{cw^T x_i}}{1+e^{-x_i^T cw}}\right) \\ \frac{\partial L(cw)}{\partial c} &= -\sum_{i=1}^{n} \frac{y_i w^T x + (y_i - 1) w^T x_i e^{cw^T x_i}}{1+e^{cw^T x_i}} \end{aligned} \tag{4}$$

   Since all of our examples are classified correctly, we know that for any score we have $x_i^T cw \geq 0$, therefore our derivative expression will always be negative, and we will always be able to minimize our NLL further.

   This makes sense as multiplying by a scalar, c, increases the likelihood function by marginal amounts as we are using the sigmoid curve which approaches (but never reaches) 1 as

$cx^T w \to \infty$ and approaches 0 (but never reaches) as $cx^T w \to -\infty$, while not adding to the accuracy of our predictions as we already have a linearly separating classifier (much like having many separating hyper-planes in linearly separable Non-Margin maximizing SVM). Therefore, MLE is not well-defined in this case.

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

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

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

   We know that the sum of two convex functions is itself a convex function. We know that any norm is a convex function, and infact, the L2 norm is strictly convex. A quick second derivative test will prove this, and the Hessian will be Positive Definite:

   $$Let\ f(x) = \lambda ||w||^2 \text{ then } \nabla f(x) = 2\lambda w \text{ and } H_{f(x)} = 2\lambda I d_n \text{ where } \lambda > 0$$

   We can then analyze the ERM part of the function, $\hat{R}_n(w)$, we note that its a summation of log terms. From the convex optimization notes, we know that a log of a summation of exponential terms is a convex function (its easier to see this when considering $1 = e^0$). Therefore, the ERM part of our objective is a summation, of logarithms of summations of exponential terms, AKA a summation of convex functions.

   Since we are summing $n + 1$ convex functions, (n in ERM and $+1$ for the regularization term) we know our regularized logistic regression objective function is indeed convex.

5. Complete the `f_objective` function in the skeleton code, which computes the objective function for $J_{\text{logistic}}(w)$. (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.

6. 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.

7. Find the ℓ_2 regularization parameter that minimizes the log-likelihood on the validation set. Plot the log-likelihood for different values of the regularization parameter.

8. [Optional] It seems reasonable to interpret the prediction $f(x) = \phi(w^T x) = 1/(1 + e^{-w^T x})$ 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 $f(x) \approx 0.7$ 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.

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

   We can show this by using the definition, then expanding the terms by substituing the definitions, and manipulate using the Chain Rule and Bayesian Probability:

$$\begin{aligned}
\theta_1 \times \theta_2 &= P(z = H|\theta_1)P(x = H|\theta_1, \theta_2) \\
&= P(z = H|\theta_1, \theta_2)P(x = H|\theta_1, \theta_2) \\
&= P(z = H|\theta_1, \theta_2)\frac{P(z = H, x = H, \theta_1, \theta_2)}{P(z = H, \theta_1, \theta_2)} \\
&= P(z = H|\theta_1, \theta_2)\frac{P(\theta_1)P(\theta_2|\theta_1)P(x = H, \theta_1, \theta_2)}{P(z = H|\theta_1, \theta_2)P(\theta_1)P(\theta_2|\theta_1)} \\
&= P(x = H|, \theta_1, \theta_2)
\end{aligned} \tag{5}$$

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.

$$\mathcal{D}_r = (\theta_1\theta_2)^{n_h} \times (1 - \theta_1\theta_2)^{n_t} = (\theta_1\theta_2)^{n_h} \times ((1 - \theta_1) + (\theta_1(1 - \theta_2))^{n_t}$$

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

    You cannot use MLE here as we have too many unknowns.

    Take for example the true value of the coin is that it always flips tails, i.e. that $\theta_1 = 0$ and that we always record the outcome correctly, that $\theta_2 = 1$. Using MLE we wouldn't know the truth: we could easily conclude that the coin always flips heads $\theta_1 = 1$ and that we always fail to report the correct result: $\theta_2 = 0$. We can further demonstrate this mathematical, by trying out MLE and taking the derivative with respect to θ_1 or θ_2 and setting to 0. Unfortunately, trying to solve this will yield nothing of value. In short, MLE fails to work here as we have too many dependent probabilities we are trying to calculate with too many degrees of freedom.

In [1]:
```
#Import necessary Libraries
import numpy as np
import scipy
from scipy.optimize import minimize
from functools import partial
import pandas as pd
import matplotlib.pyplot as plt
```

Question 5

In [2]:
```
def f_objective(theta, X, y, l2_param):
    '''
    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
    '''
    #Helper variable for ERM Summation
    objective = 0
    #Iterate over data points
    for i in range(len(y)):
        #Calculate 1+e^margin
        m = -y[i]*theta@X[i,:]
        #Add the margin exponentiation to the ERM summation
        objective += np.logaddexp(0, m)
    #Calculate the regularization penalty
    l2_penalty = l2_param* (theta@theta)
    return (objective / len(y))+ l2_penalty
```

Question 6

In [3]:
```
def fit_logistic_reg(X, y, objective_function, l2_param):
    '''
    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 object
        l2_param: regularization parameter

    Returns:
        optimal_theta: 1D numpy array of size num_features
    '''
    starting_theta = np.ones(X.shape[1])
    passing_func = partial(objective_function,X=X,y=y,l2_param=l2_para
    optimal_theta = minimize(passing_func,starting_theta).x
    return optimal_theta
```

In [4]:
```
#Load Data, then Clean it
X_train, X_val = pd.read_csv('X_train.txt',header=None), pd.read_csv('
y_train, y_val = pd.read_csv('y_train.txt',header=None), pd.read_csv('
#Fix y_train / val labels to -1 rather than 0
y_train[y_train==0], y_val[y_val==0] = -1, -1
#Standardize via Z-Scores, Add Bias Column
X_val = (X_val - X_train.mean() )/ X_train.std()
X_train = (X_train - X_train.mean() )/ X_train.std()
#X_val = (X_val - X_val.mean())/ X_val.std()
X_train['Bias'] = 1
X_val['Bias'] = 1

#Transform DataFrames to numpy ararys
X_train = X_train.to_numpy()
X_val = X_val.to_numpy()
y_train = y_train.to_numpy()
y_val = y_val.to_numpy()
```

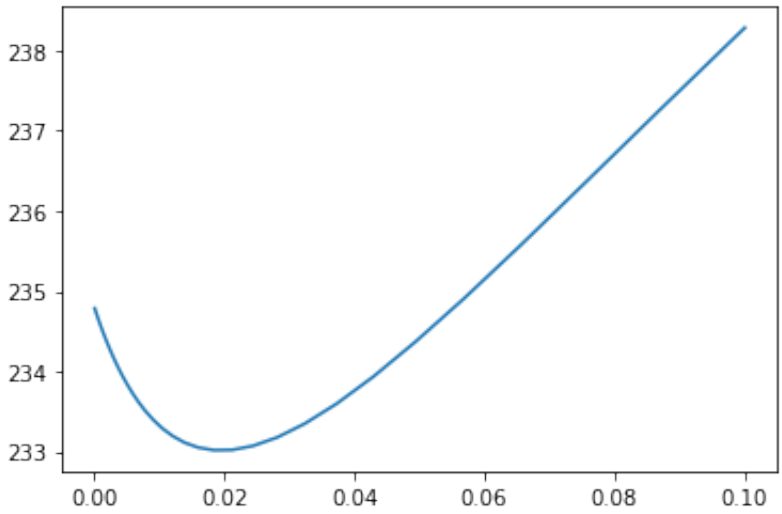
Problem 7

In [5]:
```
def NLL(X,y,theta):
    nll = 0
    for i in range(len(y)):
        m = -y[i]*theta@X[i,:]
        nll += np.logaddexp(0,m)
    return(nll)
```

In [6]:
```
#lambda_list = np.linspace(0,2,10)
lambda_list = np.logspace(-4,-1,num=50)

nll_list = []
for lamda_reg in lambda_list:
    optimal_theta = fit_logistic_reg(X_train, y_train,f_objective, l2_
    nll_list.append(NLL(X=X_val,y=y_val,theta=optimal_theta))
```

In [7]:
```
plt.plot(lambda_list,nll_list)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x7fd0119da700>]
```



In [8]:
```
min_nll = min(nll_list)
min_index = nll_list.index(min_nll)
min_lambda = lambda_list[min_index]
print("Min NLL:",min_nll,"Lambda:",min_lambda)
```

```
Min NLL: 233.0183069467581 Lambda: 0.018420699693267165
```

Minimum NLL achieved at $\lambda = 0.01842$, with $NLL = 233$

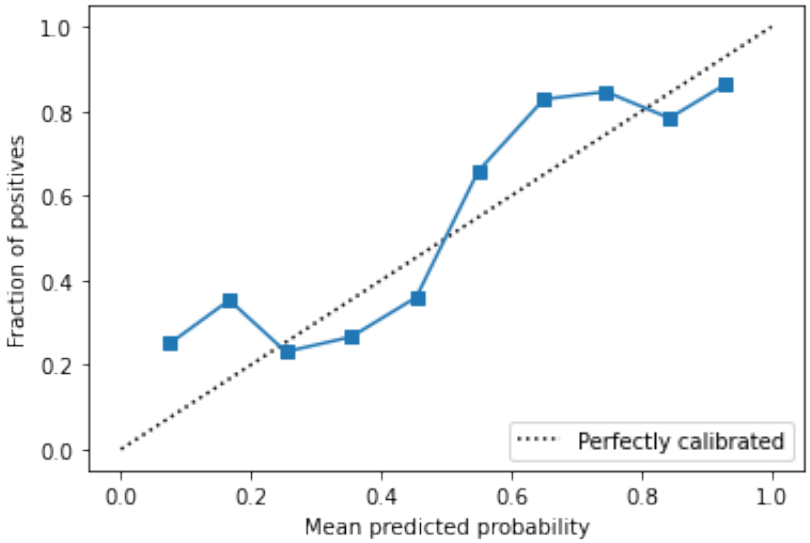
Problem 8

In [9]:
```
#Use the Optimal Theta Calculated in the Last Problem
optimal_theta = fit_logistic_reg(X_train,y_train,f_objective,l2_param=
```

In [10]:
```
#Initialize helper variable
scores = []
#Calculate a score for each observation in X_val
for i in range(len(X_val)):
    score = 1 / (1 + np.exp(-optimal_theta@X_val[i,:]))
    scores.append(score)
```

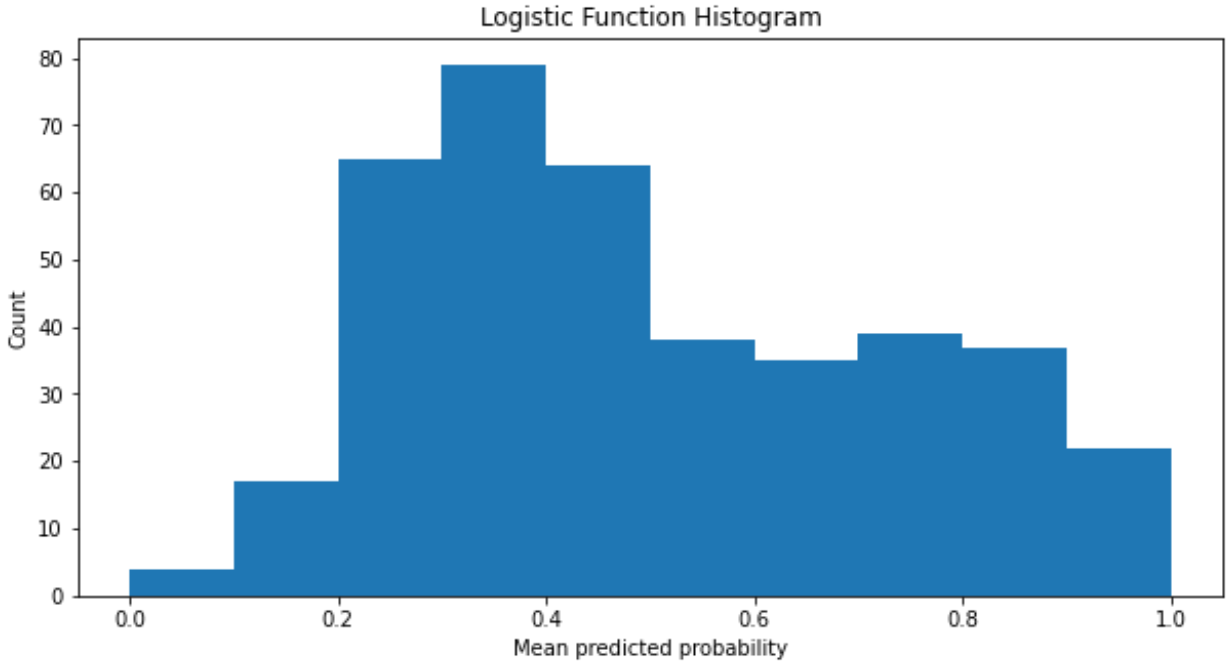
In [11]:
```
#Use Sci-Kit Learn Template to Show how Calibrated Our Model is
from sklearn.calibration import calibration_curve, CalibrationDisplay
y_true, y_prob = y_val, scores

prob_true, prob_pred = calibration_curve(y_true, y_prob, n_bins=10)
disp = CalibrationDisplay(prob_true, prob_pred,y_prob)
disp.plot()
```

Out[11]: <sklearn.calibration.CalibrationDisplay at 0x7fd002044850>



In [12]:
```
plt.figure(figsize=(10,5))
plt.hist(disp.y_prob, range=(0,1), bins=10, histtype='bar')
plt.title('Logistic Function Histogram')
plt.xlabel("Mean predicted probability")
plt.ylabel("Count")
```

Out[12]:
```
Text(0, 0.5, 'Count')
```



Commentary:

It appears that our model is not very well calibrated: using a bin size of 10, we can clearly see that the actual outcome percentage of positive results does not match the predicted probability. Visually, we can see this occuring as if our model was calibrated perfectly, we would have the points on our plot line up with the linear line $y = x$.

Interpolating further, our model is not calibrated well, as if we have a set of feature vectors $x_i \in X$, where $x_i^T w = .7$, we would expect about 70% of the labels to be positive, that is, $y = 1$. However, using our plot, we can see that the actual mix of positive and negative ground truth labels does not match our predicted 70%, and is actually 80%.

Furthermore, by analyzing the distribution of scores $x^T w$, we see many predictions that do not display the most confidence, or margin. This is apparant in the histrogram with bin size = 10, as we can see many of our predicted scores fall between the range $0.2 \leq x^T w \leq 0.6$, where as if our model was calibrated well, the majority of scores would fall in the $0.0 - 0.1$ bucket, and many others in the $0.9 - 1.0$ bucket. Since we used the lambda function that would minimize our NLL, I don't think the issue is caused by our model selection, but perhaps that the data we are analyzing is not well predicted by a logistic regression model.