logistic-regression

April 5, 2019

```
In [102]: import numpy as np
          import pandas as pd
          from sklearn.preprocessing import StandardScaler
          import matplotlib.pyplot as plt
          from sklearn.linear_model import LogisticRegression
          from sklearn.calibration import calibration_curve
```

```
In [103]: from logreg_skeleton import fit_logistic_reg, f_objective
```

3.3 Regularized Logistic Regression

3.3.1

Prove that the objective function is convex.

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

LogSumExp is convex, therefore $\log\left(1+\exp\left(-y_i w^T x_i\right)\right)$ is convex.

Sum of convex functions is convex, therefore $\sum_{i=1}^{n}\log\left(1+\exp\left(-y_i w^T x_i\right)\right)$ is convex.

Dividing a convex function by a positive constant is convex, therefore $\frac{1}{n}\sum_{i=1}^{n}\log\left(1+\exp\left(-y_i w^T x_i\right)\right)$ is convex.

Norms are convex, therefore $\|w\|^2$ is convex.

Multiplying a convex function by a positive constant is convex, therefore $\lambda\|w\|^2$ is convex.

Thus $J_{\text{logistic}}(w) = \frac{1}{n}\sum_{i=1}^{n}\log\left(1+\exp\left(-y_i w^T x_i\right)\right) + \lambda\|w\|^2$ is convex.

3.3.2

Complete the f_objective function

```
In [101]: 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 = len(y)
              pred = X@theta
              margin = -y*pred
              log_loss = np.logaddexp(0, margin).sum()
              reg = l2_param*(theta@theta)
              return log_loss/n + reg
```

3.3.3

Complete the fit_logistic_reg function and use it to train a model on the procided data.

```
In [19]: 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
             '''
             objective_function = partial(objective_function, X=X, y=y, l2_param=l2_param)

             n_features = X.shape[1]
             theta_0 = np.zeros(n_features)
             theta = minimize(objective_function, theta_0).x
             return theta
```

```
In [104]: # Load the data
          x_train = np.loadtxt('X_train.txt', delimiter=',')
          x_val = np.loadtxt('X_val.txt', delimiter=',')
          y_train = np.loadtxt('y_train.txt', delimiter=',')
          y_val = np.loadtxt('y_val.txt', delimiter=',')
```

```
In [105]: # Standardize data
          ss = StandardScaler()
          x_train = ss.fit_transform(x_train)
          x_val = ss.transform(x_val)
          y_train[y_train==0] = -1
          y_val[y_val==0] = -1
```

```
In [106]: # Add bias term
          x_train = np.append(10*np.ones((len(x_train),1)), x_train, axis=1)
          x_val = np.append(10*np.ones((len(x_val),1)), x_val, axis=1)
```

```
In [107]: # Train model
          theta = fit_logistic_reg(x_train, y_train, f_objective)
```

```
In [108]: theta
```

```
Out[108]: array([ 0.00236204,  0.00095657, -0.00030132,  0.00302058,  0.10533832,
                 -0.00358714, -0.00135921, -0.00385466, -0.00079028, -0.0011443 ,
                 -0.07179551,  0.00655072, -0.004512  ,  0.01125831, -0.003866  ,
                 -0.00271356,  0.00150264, -0.00278385, -0.00919238, -0.00682348,
                 -0.01027393])
```

3.3.4

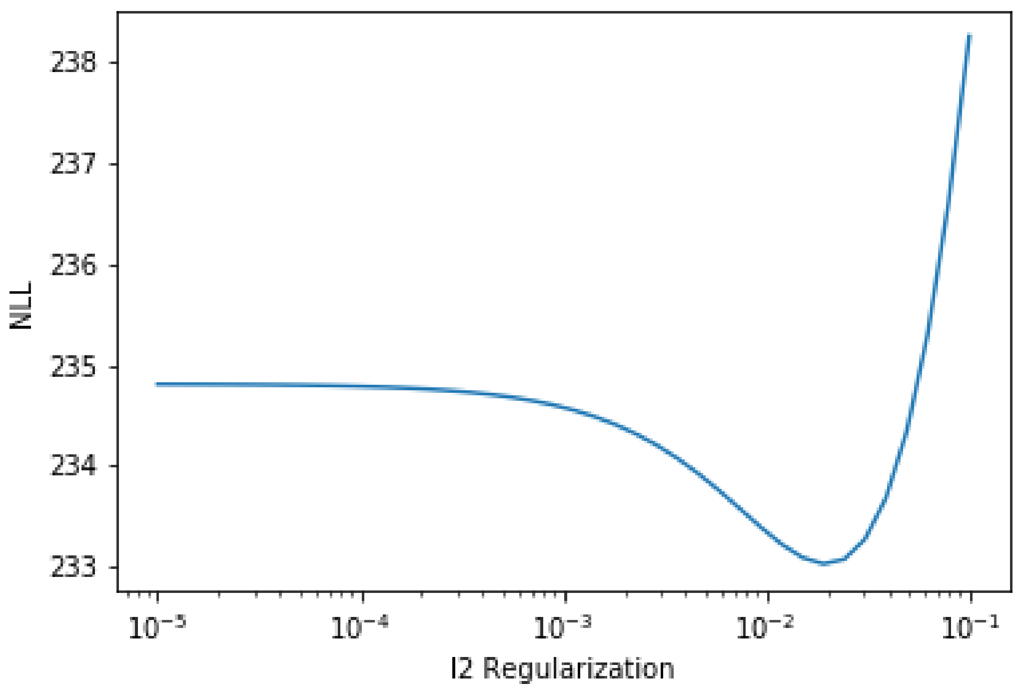
Find the l2 regulrization term that minimizes the log-likelihood on the validation set. Plot the log-likelihood for different values.

In [113]:
```
def NLL(x, y, theta):
    margin = -y * (x@theta)
    return np.logaddexp(0, margin).sum()
```

In [122]:
```
nlls = []
l2s = np.logspace(-5, -1, 40)
for l2 in l2s:
    theta = fit_logistic_reg(x_train, y_train, f_objective, l2)
    nlls.append(NLL(x_val, y_val, theta))
```

In [123]:
```
plt.plot(l2s, nlls)
plt.xscale('log')
plt.xlabel('l2 Regularization')
plt.ylabel('NLL')
```

Out[123]:
```
Text(0, 0.5, 'NLL')
```



Based on this, I'll choose 0.011 as my l2 reg

3.3.5

Calibration plot

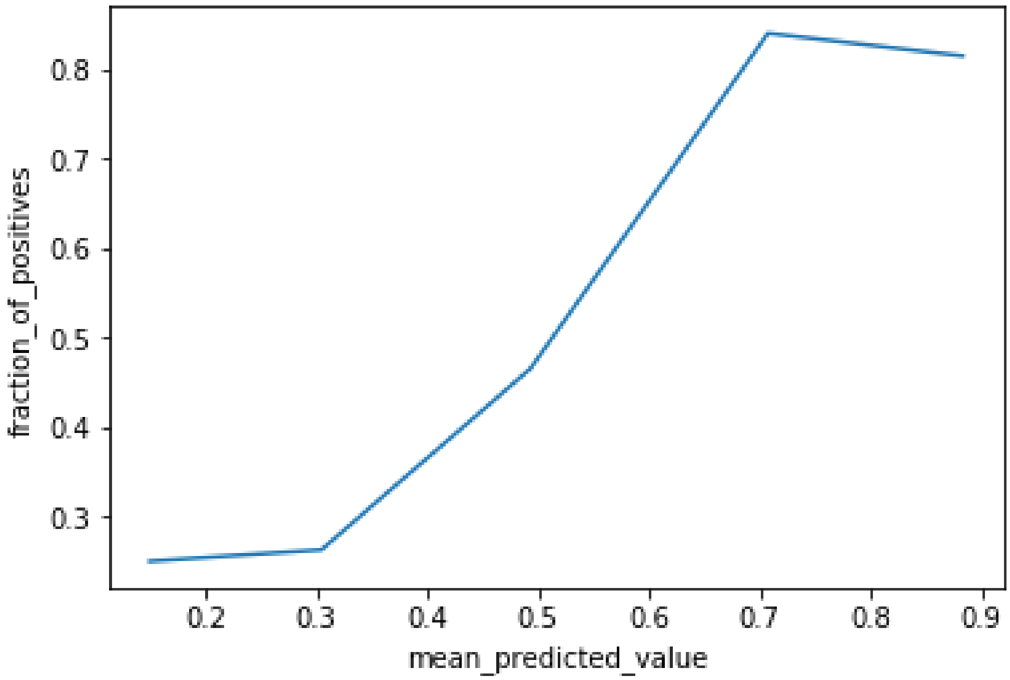
```
In [124]: def sigmoid(x):
              return 1/(1+np.exp(-x))

In [125]: theta = fit_logistic_reg(x_train, y_train, f_objective, 0.011)
          y_pred = sigmoid(x_val@theta)

In [146]: fraction_of_positives, mean_predicted_value = calibration_curve(y_val, y_pred, n_bins=

In [147]: plt.plot(mean_predicted_value, fraction_of_positives)
          plt.xlabel('mean_predicted_value')
          plt.ylabel('fraction_of_positives')

Out[147]: Text(0, 0.5, 'fraction_of_positives')
```



It appears that the model unerconfident as hown by the sigmoid shape of the calibration curve. Sigmoid calibration may help correct and calibrate the predictions.