

MA506 Probability and Statistical Inference

Lec 25: Multiclass probabilistic classification

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

Multinomial logistic Regression

Recalling, for a 2 class problem, assuming class labels are 0 and 1, the probability of class 1 was quantified by the sigmoid function:

$$P(y = 1|x) = h(x) = \frac{1}{1 + e^{-X\beta}}$$

Now in a multiclass problem, instead of class label $\{0, 1\}$ assuming we have K class labels. Hence y can take K possible values:

$$y = \{1, 2, \dots, K\}$$

For this case multinomial logistic regression assigns each class its own β_i vector. Hence:

$$P(y = i|x) = \frac{e^{X\beta_i}}{\sum_{j=1}^K e^{X\beta_j}}$$

For example, for a 3 class problem, the procedure can be summarized as follows:

1. Fit 3 logistic regression models
 - For class 1 vs all other classes (obtain weight β_1)
 - For class 2 vs all other classes (obtain weight β_2)
 - For class 3 vs all other classes (obtain weight β_3)
2. During prediction of class for a new sample x , compute:

$$P(y = 1) = \frac{e^{X\beta_1}}{e^{X\beta_1} + e^{X\beta_2} + e^{X\beta_3}}$$

$$P(y = 2) = \frac{e^{X\beta_2}}{e^{X\beta_1} + e^{X\beta_2} + e^{X\beta_3}}$$

$$P(y = 3) = \frac{e^{X\beta_3}}{e^{X\beta_1} + e^{X\beta_2} + e^{X\beta_3}}$$

3. Final class prediction is the class with highest probability:

$$i = \arg \max_i P(y = i)$$

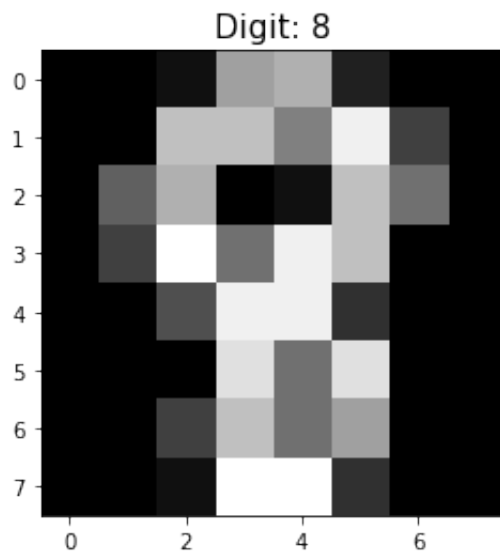
Datasets we will use

```
In [2]: from sklearn.datasets import load_digits
```

```
In [3]: digits = load_digits()  
X = digits.data  
y = digits.target
```

```
In [4]: def plot_digits(X,y):  
    index = np.random.randint(X.shape[0])  
    sample = X[index,:]  
    label = y[index]  
    plt.imshow(sample.reshape(8,8),cmap = 'gray')  
    plt.title(f'Digit: {label}',size =15)  
    plt.show()
```

```
In [5]: plot_digits(X,y)
```



Dividing data into training and testing

```
In [6]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,
```

```
In [7]: set(y)
```

```
Out[7]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Logistic Regression

```

In [8]: def h(X,beta):
        ## returns the value of the sigmoid function
        ypred = 1/(1 + np.exp(-1*X.dot(beta)))
        return ypred

def model_cost(X,y,beta):
    ## computes the value of the model fitting cost
    cost = 0
    for j in range(X.shape[0]):
        term1 = y[j]*np.log(h(X[j,:],beta))[0]
        term2 = (1 - y[j])*np.log(1 - h(X[j,:],beta))[0]
        cost = cost + (term1+term2)
    return -1*cost

def gradient_descent(X,y,beta0,alpha,iteration):
    ## Implemented iteration for the gradient descent algorithm
    ypred = h(X,beta0)
    beta = [beta0]
    cost = [model_cost(X,y,beta0)]
    for j in range(iteration):
        grad_l = -1*X.T.dot(y.reshape(-1,1) - ypred)
        beta_new = beta[-1] - alpha * grad_l
        ypred = h(X,beta_new)
        beta.append(beta_new)
        #print(f'beta: {beta_new[0][0],beta_new[1][0]}')
        print('Cost at iteration '+str(j)+' is: ',model_cost(X,y,beta_
        cost.append(model_cost(X,y,beta_new))
    return [beta_new,cost]

```

```

In [9]: classes = list(set(y))
        classes

```

```

Out[9]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

```

In [10]: ## Number of classes
         n_classes = len(classes)
         n_classes

```

```

Out[10]: 10

```

In [11]: `X_train[0,:]`

Out[11]: `array([0., 0., 0., 8., 10., 14., 3., 0., 0., 1., 13., 13., 9
,
 12., 8., 0., 0., 6., 16., 8., 8., 16., 4., 0., 0., 5
,
 16., 16., 16., 9., 0., 0., 0., 0., 5., 8., 14., 12., 0
,
 0., 0., 0., 0., 3., 16., 5., 0., 0., 0., 0., 0., 15
,
 8., 0., 0., 0., 0., 0., 1., 12., 2., 0., 0., 0.])`

In [12]: `from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_train[0,:]`

Out[12]: `array([0. , -0.34338378, -1.09642621, -0.88717562, -0.4282557
1,
 1.43845406, 0.49363934, -0.12501886, -0.05963551, -0.3151569
,
 0.47849512, 0.21918793, -0.28691819, 0.6197714 , 1.7692282
5,
 -0.12614188, -0.03992024, 0.96822696, 1.0921471 , 0.1475735
1,
 0.14198996, 1.32778473, 0.70793513, -0.11294863, -0.0282166
3,
 0.80906081, 1.10876452, 1.21974443, 0.98975536, 0.2628211
4,
 -0.64074225, -0.03992024, 0. , -0.67349682, -0.4136552
2,
 -0.16812017, 0.62783539, 0.5664439 , -0.82668859, 0.
,
 -0.06209797, -0.53059473, -1.06319169, -0.66713857, 1.3181608
7,
 -0.55060546, -0.7726095 , -0.09516311, -0.04237744, -0.4087359
5,
 -1.33984381, 1.01453563, -0.30892748, -1.43653873, -0.7348467
,
 -0.19859912, -0.02821663, -0.30739208, -0.90153224, -0.0186647
7,
 -1.98373642, -1.12193961, -0.48894777, -0.18556033])`

In []:

```
In [13]: beta0 = 0.00001*np.ones([X.shape[1]+1,1])
alpha = 0.01
iteration = 100
beta_vec = []
Xs = np.concatenate((np.ones([len(X_train),1]),X_train),axis = 1)
for i in classes:
    y_new = y_train.copy()
    y_new[y_train == i] = 1
    y_new[y_train != i] = 0
    beta_new,cost = gradient_descent(Xs,y_new,beta0,alpha,iteration)
    beta_vec.append(beta_new)

/var/folders/d7/n2s59n5f38lzc7727n300000gn/T/ipykernel_91179/3530693907.py:11: RuntimeWarning: invalid value encountered in multiply
    term2 = (1 - y[j])*np.log(1 - h(X[j,:],beta))[0]

Cost at iteration 0 is: nan
Cost at iteration 1 is: 549.0099941540459
Cost at iteration 2 is: 288.0617736451865
Cost at iteration 3 is: 82.06124404147252
Cost at iteration 4 is: 21.629639743589912
Cost at iteration 5 is: 15.22731254112984
Cost at iteration 6 is: 12.288884859943476
Cost at iteration 7 is: 10.558215412713686
Cost at iteration 8 is: 9.378177526046562
Cost at iteration 9 is: 8.49551859151884
Cost at iteration 10 is: 7.797559779439314
Cost at iteration 11 is: 7.227066700260707
Cost at iteration 12 is: 6.750835895196942
Cost at iteration 13 is: 6.347297156958587
Cost at iteration 14 is: 6.001309603976792
Cost at iteration 15 is: 5.701750854901429
```

```
In [14]: beta_vec[4]
```

```
Out[14]: array([[ -9.93304597e+00],
 [ 1.00000000e-05],
 [-5.38921268e-01],
 [-1.60323415e-01],
 [-1.81609413e+00],
 [-8.90686307e-01],
 [-1.47830750e+00],
 [-7.98005142e-01],
 [-1.32727588e-01],
 [ 1.59802758e-01],
 [ 4.43687047e-01],
 [-6.67507736e-01],
 [-9.55875932e-01],
 [-1.27129960e+00],
 [-1.03782760e+00],
 [-5.75373193e-01]])
```

```

[ 0.75575155e-01],
[ 1.38021867e-01],

[ 1.03603834e-01],
[ 1.13704450e-01],
[ 8.18319305e-01],
[ 2.32574703e-01],
[-2.61597733e-01],
[ 6.68781947e-01],
[-2.40264735e-01],
[ 3.89255084e-01],
[ 4.57608029e-01],
[-2.73849265e-01],
[ 8.84637448e-01],
[-2.27743491e-01],
[-3.18854238e-02],
[ 3.61563421e-01],
[ 1.09251728e+00],
[ 6.47350912e-01],
[ 1.00000000e-05],
[ 7.98969647e-01],
[-2.61998626e-01],
[-1.61799333e-01],
[ 6.79581219e-01],
[ 3.70962746e-01],
[ 8.38080701e-01],
[ 1.00000000e-05],
[ 1.00705269e+00],
[ 1.16067477e+00],
[-7.52213587e-01],
[ 1.71005494e+00],
[ 9.99215651e-01],
[ 1.98261336e-01],
[-1.81196184e-01],
[ 1.13329558e-01],
[ 4.34855670e-01],
[ 9.61206064e-01],
[-9.38920947e-01],
[ 2.50082836e-02],
[ 2.52531401e-01],
[-1.14570015e+00],
[-9.20970999e-01],
[ 1.75965040e-01],
[-2.70501343e-01],
[-5.34365588e-01],
[-1.22850338e-01],
[-3.33585184e-01],
[-9.56462336e-02],
[-1.00405609e+00],
[-3.14041548e-01],
[-7.82933766e-03]])

```

In []:

```
In [15]: def pred(X,beta_vec):
    prob1 = []
    for i in X:
        temp = []
        for j in range(10):
            beta = beta_vec[j]
            prob = np.exp(i.dot(beta))
            temp.append(prob)
        prob1.append([ii/sum(temp) for ii in temp])
    probability = np.array(prob1)
    return probability.reshape(len(X),len(beta_vec))
```

In [16]: X_test[:,10]

```
Out[16]: array([16., 14., 15.,  9., 10.,  4.,  0., 11., 14., 16., 14., 16.,  5
'',
        5.,  1.,  6.,  0., 14., 16., 15., 16.,  9., 13., 14., 16., 16
'',
        6., 13., 15., 14.,  8., 14.,  6.,  9., 13.,  0., 15.,  5., 13
'',
        0., 12., 12., 13., 14.,  8., 10.,  4., 12., 13.,  0., 13.,  0
'',
        7., 16., 12., 15., 15., 13.,  9., 10., 10., 14., 14.,  5.,  2
'',
        12.,  9.,  4., 15.,  6., 15., 15., 14.,  0., 15.,  7., 14., 11
'',
        15., 15., 12., 15.,  1., 16., 13., 16.,  0., 16.,  7., 15., 14
'',
        7., 12.,  0.,  8.,  6.,  0.,  9., 16.,  5., 16., 16., 15., 10
'',
        13., 10.,  7.,  0.,  1.,  4.,  4., 14., 14., 13.,  9., 15., 16
'',
        8.,  0., 14., 13., 15., 16.,  1.,  6.,  8., 13.,  6.,  2., 15
'',
        16.,  4., 14., 16., 12., 16., 16., 15., 16.,  7., 16., 14.,  3
'',
        9., 10.,  0., 16., 13., 12., 12., 14.,  0., 16., 12.,  4.,  2
'',
        16.,  0.,  2., 14.,  1., 10., 11., 16., 11., 11., 15., 12., 14
'',
        13.,  3.,  0., 13., 12.,  7., 12., 12., 11., 13.,  6.,  2., 14
'',
        11., 16., 16., 14., 16., 11., 16.,  2.,  8., 15., 12., 16., 16
'',
        14., 14., 12., 13.,  3., 16., 11.,  2.,  8.,  4.,  1., 16., 16
'',
```



```
13., 15., 16., 11., 6., 4., 3., 0., 0., 2., 3., 16., 16
'',
0., 16., 16., 2., 8., 6., 14., 14., 5., 7., 16., 16., 9
'',
16., 9., 16., 14., 9., 9., 0., 12., 16., 10., 12., 11., 7
'',
14., 16., 6., 15., 16., 14., 12., 16., 13., 12., 16., 10., 7
'',
15., 9., 9., 12., 15., 14., 0., 10., 16., 15., 12., 1., 9
'',
13., 12., 8., 1., 5., 16., 16., 2., 9., 1., 15., 16., 0
'',
16., 13., 16., 0., 11., 16., 0., 16., 5., 8., 2., 0., 16
'',
15., 16., 10., 9., 15., 14., 6., 8., 10., 2., 16., 16., 10
'',
16., 16., 7., 10., 15., 15., 14., 7., 9., 10., 16., 10., 13
'',
0., 3., 0., 13., 16., 15., 14., 12., 10., 10., 13., 16., 16
'',
16., 9., 14., 7., 9., 1., 15., 14., 12., 6., 0., 4., 10
'',
9., 16., 15., 6., 13., 5., 7., 0., 9., 5., 13., 14., 6
'',
7., 0., 15., 6., 9., 15., 4., 7., 16., 16., 16., 16., 15
'',
16., 16., 13., 1., 8., 4., 14., 8., 0., 8., 15., 12., 6
'',
13., 16., 15., 5., 5., 16., 15., 9., 8., 16., 13., 5., 15
'',
14., 11., 13., 16., 6., 1., 14., 16., 16., 14., 16., 14., 16
'',
4., 16., 15., 12., 0., 16., 7., 14., 12., 0., 11., 7., 16
'',
16., 14., 15., 5., 6., 0., 11., 16., 16., 15., 9., 16., 8
'',
4., 3., 0., 13., 16., 16., 12., 15., 15., 15., 16., 7., 15
'',
11., 0., 16., 16., 16., 7., 16., 14., 14., 8., 16., 16., 11
'',
16., 12., 14., 15., 13., 11., 1., 0., 15., 8., 0., 16., 0
'',
14., 10., 2., 8., 14., 14., 15., 0., 15., 3., 12., 16., 6
'',
13., 10., 5., 15., 16., 11., 16., 5., 9., 0., 16., 0., 12
'',
14., 6., 12., 11., 10., 3., 16., 8., 0., 0., 10., 16., 12
'',
15., 0., 16., 12., 16., 5., 6., 15., 15., 3., 8., 16., 10
'',
```

```
8., 0., 8., 15., 11., 10., 13.]])
```

```
In [17]: X_test = scaler.transform(X_test)
X_test[:,10]
```

```
Out[17]: array([ 1.02677988,  0.66125671,  0.84401829, -0.25255121, -0.0697896
3,
-1.16635913, -1.89740546,  0.11297196,  0.66125671,  1.0267798
8,
0.66125671,  1.02677988, -0.98359754, -0.98359754, -1.7146438
8,
-0.80083596, -1.89740546,  0.66125671,  1.02677988,  0.8440182
9,
1.02677988, -0.25255121,  0.47849512,  0.66125671,  1.0267798
8,
1.02677988, -0.80083596,  0.47849512,  0.84401829,  0.6612567
1,
-0.43531279,  0.66125671, -0.80083596, -0.25255121,  0.4784951
2,
-1.89740546,  0.84401829, -0.98359754,  0.47849512, -1.8974054
6,
0.29573354,  0.29573354,  0.47849512,  0.66125671, -0.4353127
9,
-0.06978963, -1.16635913,  0.29573354,  0.47849512, -1.8974054
6])
```

```
In [18]: Xtest = np.concatenate((np.ones([len(X_test),1]),X_test),axis = 1)
prob = pred(Xtest,beta_vec)
```

```
In [19]: prob.shape
```

```
Out[19]: (540, 10)
```

```
In [20]: prob[0,:]
```

```
Out[20]: array([1.34388604e-10, 3.92410348e-12, 9.99999987e-01, 5.99393885e-13
,
2.30976318e-12, 6.15516496e-14, 1.34137578e-20, 1.31930729e-08
,
4.43311739e-11, 1.51339498e-12])
```

```
In [21]: ind = np.argmax(prob,axis = 1)
```

```
In [22]: ind[0]
```

```
Out[22]: 2
```

```
In [23]:
```

ind

```
Out[23]: array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5, 2, 8, 6, 6, 6, 6, 1, 0, 5, 8, 8,
7,
      8, 4, 7, 5, 4, 9, 2, 9, 4, 7, 6, 8, 9, 4, 3, 8, 0, 1, 8, 6, 7,
7,
      1, 0, 7, 6, 2, 1, 9, 6, 7, 9, 0, 0, 5, 1, 6, 3, 0, 2, 3, 4, 1,
9,
      2, 6, 9, 1, 8, 3, 5, 1, 2, 1, 2, 2, 9, 7, 2, 3, 6, 0, 5, 3, 7,
5,
      1, 2, 9, 9, 3, 1, 7, 7, 4, 8, 5, 8, 5, 5, 2, 5, 9, 0, 7, 1, 4,
7,
      3, 4, 8, 9, 7, 9, 8, 2, 1, 5, 2, 5, 8, 4, 1, 7, 0, 6, 1, 5, 5,
9,
      9, 5, 9, 9, 5, 7, 5, 6, 2, 8, 6, 7, 6, 1, 5, 1, 5, 9, 9, 1, 5,
3,
      6, 1, 8, 9, 8, 7, 6, 7, 6, 5, 6, 0, 8, 8, 9, 8, 6, 1, 0, 4, 1,
6,
      3, 8, 6, 7, 4, 1, 6, 3, 0, 3, 3, 3, 0, 7, 7, 5, 7, 8, 0, 7, 1,
9,
      6, 4, 5, 0, 1, 4, 6, 4, 3, 3, 0, 9, 5, 9, 2, 1, 4, 2, 1, 6, 8,
9,
      2, 4, 9, 3, 7, 6, 2, 3, 3, 1, 6, 9, 3, 6, 3, 2, 2, 0, 7, 6, 1,
1,
      9, 7, 2, 7, 8, 5, 5, 7, 5, 3, 3, 7, 2, 7, 5, 5, 7, 0, 9, 1, 6,
5,
      9, 7, 4, 3, 8, 0, 3, 6, 4, 6, 3, 2, 6, 8, 8, 8, 4, 6, 7, 5, 2,
4,
      5, 3, 2, 4, 6, 9, 4, 5, 4, 3, 4, 6, 2, 9, 0, 6, 7, 2, 0, 9, 6,
0,
      4, 2, 0, 7, 9, 8, 5, 7, 8, 2, 8, 4, 3, 7, 2, 6, 9, 1, 5, 1, 0,
8,
      2, 4, 9, 5, 6, 2, 2, 7, 2, 1, 5, 1, 6, 4, 5, 0, 9, 4, 1, 1, 7,
0,
      8, 9, 0, 5, 4, 3, 8, 8, 6, 5, 3, 4, 4, 4, 8, 8, 7, 0, 9, 6, 3,
5,
      2, 3, 0, 8, 2, 3, 1, 3, 3, 0, 0, 4, 6, 0, 7, 7, 6, 2, 0, 4, 4,
2,
      3, 7, 1, 9, 8, 6, 8, 5, 6, 2, 2, 3, 1, 7, 7, 8, 0, 3, 3, 2, 1,
5,
      5, 9, 1, 3, 7, 0, 0, 3, 0, 4, 5, 9, 3, 3, 4, 3, 1, 8, 9, 8, 3,
6,
      2, 1, 6, 2, 1, 7, 5, 5, 1, 9, 2, 9, 9, 7, 2, 1, 4, 9, 3, 2, 6,
2,
      5, 9, 6, 5, 8, 2, 0, 7, 8, 0, 6, 8, 4, 1, 8, 6, 4, 3, 4, 2, 0,
4,
      5, 8, 3, 9, 1, 8, 3, 4, 5, 0, 8, 5, 6, 3, 0, 6, 9, 1, 5, 2, 2,
8,
      9, 8, 4, 2, 3, 0, 7, 8, 8, 1, 1, 3, 5, 5, 8, 4, 9, 7, 8, 4, 4,
9,
```

```
0, 1, 6, 9, 3, 6, 1, 7, 0, 6, 2, 9])
```

```
In [24]: from sklearn.metrics import accuracy_score
```

```
In [25]: accuracy_score(y_test,ind)
```

```
Out[25]: 0.9629629629629629
```

```
In [ ]:
```