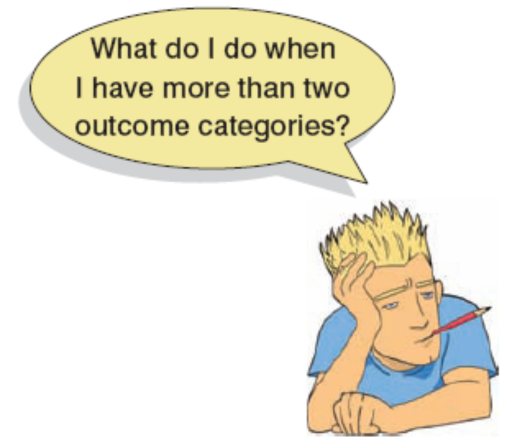
Multi-class Logistic Regression

Mengdi Huai

11/08/2017

Aims



- Suppose there are N M-dimensional training data
  - N: the number of training
  - M: dimensions
- Our goal is to classify into one of K multiple exclusive categories (i.e., class)
  - e.g. K={1,2,3,…,K}

Multi-class Logistic Regression

- For K classes, we work with soft-max function instead of logistic sigmoid.
  - $p(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$
  - where $a_k = \boldsymbol{w}_k^T \phi + b_k$, k=1,2,3,…,K
  - ϕ denotes the feature vectors
  - $\boldsymbol{w}_k = [w_{k1}, w_{k2}, \dots, w_{kM}]^T$ and $a = \{a_1, \dots, a_K\}$
  - We need to learn a set of K weight vectors {$\boldsymbol{w}$

Objective Function

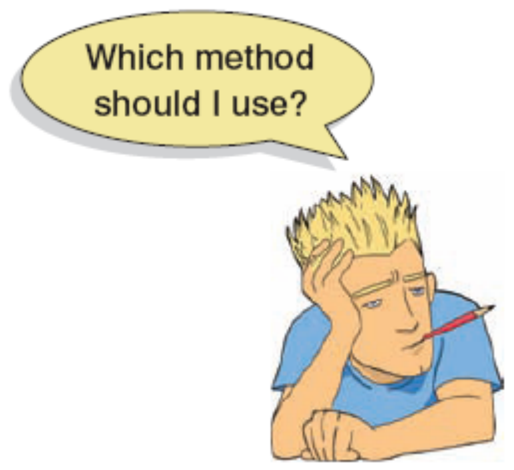
- We use maximum likelihood to determine the parameters $\{w_1, w_2, \ldots, w_K\}$
  - N: the number of classes
- Objective Function: negative log-likehood

  Cross-entropy error

$$E(w_1, \ldots, w_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} ln y_{nk}.$$

  Where t_{nk} corresponds to sample *n* and class *k*. And, $y_{nk} = y_k(\phi_n) = \frac{\exp(\boldsymbol{w}_k^T \phi_n)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_k^T \phi_n)}$, where ϕ_n denotes the *n*-th feature vector.

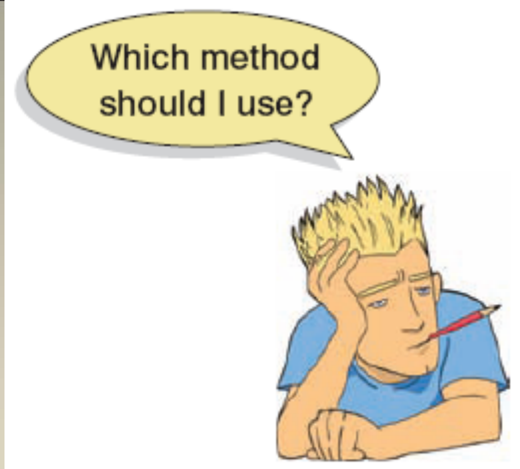
Gradient



- Gradient of the above error function wrt parameter $\boldsymbol{w}_k$

$$\nabla_{w_k} E(w_1, \ldots, w_K) = -\sum_{n=1}^{N} (y_{nk} - t_{nk})\, \phi_n$$

Where t_{nk} corresponds to sample *n* and class *k*. And, $y_{nk} = y_k(\phi_n) = \frac{\exp(\boldsymbol{w}_k^T \phi_n)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_k^T \phi_n)}$, where ϕ_n denotes the *n*-th feature vector.



Gradient Descent

- We can use the sequential algorithm in which inputs are presented one at a time in which the weight vector is updated using

$$w_k^{\tau+1} = w_k^{\tau} - \eta * \nabla_{w_k} E(w_1, \ldots, w_K)$$

where τ denotes the iteration number and η denotes the size of step to be used for gradient descent.

Pseudocode

1. Initialize w_j^0, j=1,2,3,…,K
2. for τ from 1 to IteNum
3. for k from 1 to K
4. $w_k^{\tau+1}=w_k^{\tau} - \eta * \nabla_{w_k} E(w_1, \ldots, w_K)$
5. endfor
6. endfor

***Note that IteNum denotes the number of iterations.**

Experiments

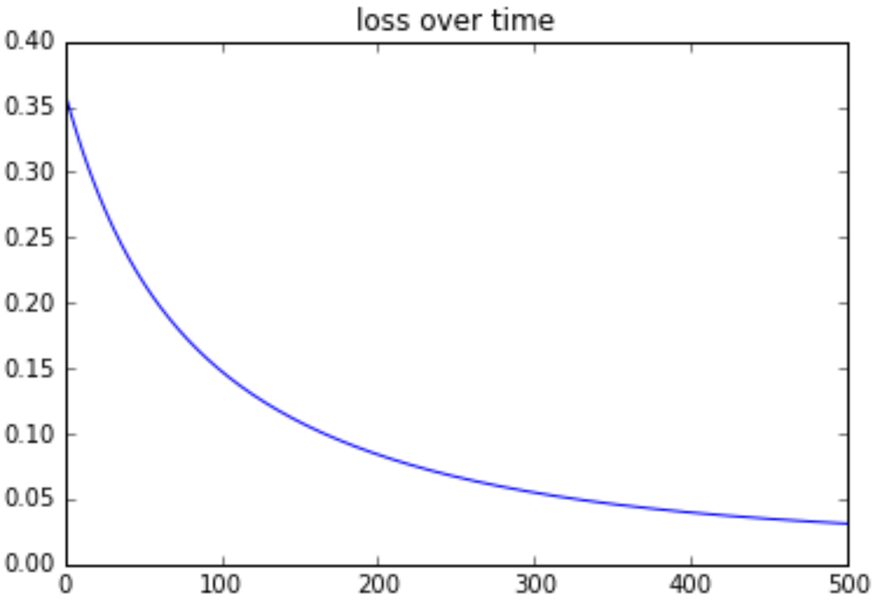
- Here, we use sklearn.datasets.make_classification() to generate 3 datasets, each of which contains 100 2-dimensional data points with 3 classes.

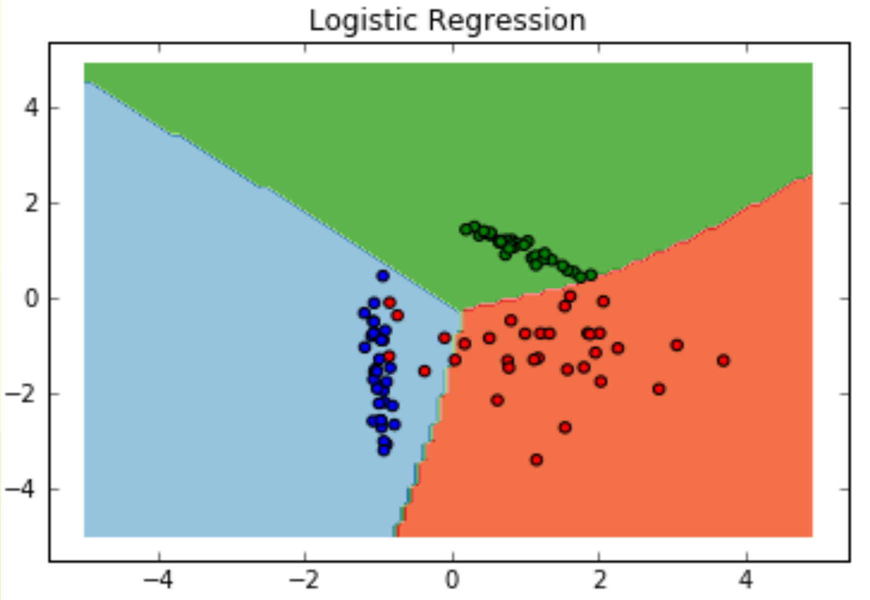
```
# #make data
X,Y = make_classification(n_features=2, n_informative=2, n_redundant=0,
                          n_repeated=0, n_classes=3, n_clusters_per_class=1)
```

- In the following, we give some experiments results.
  - The error cost over iterations
  - How well the final model discriminates the data points

Dataset 1

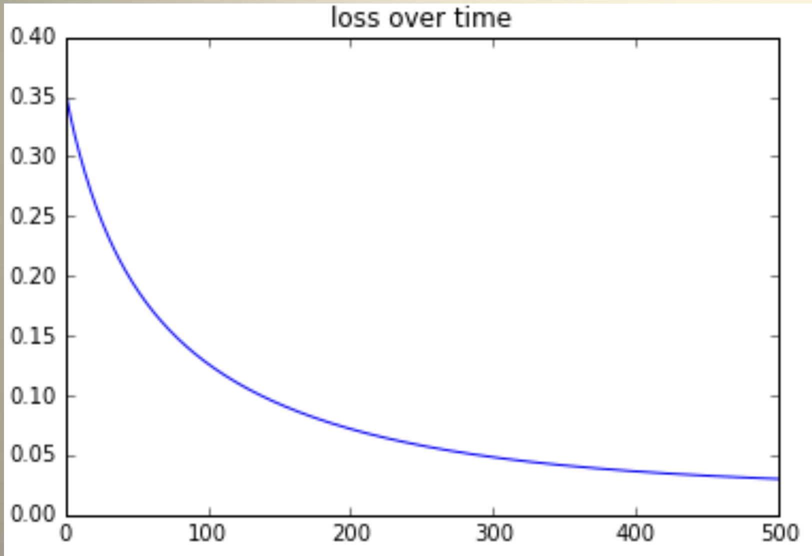
- Loss and Boundaries

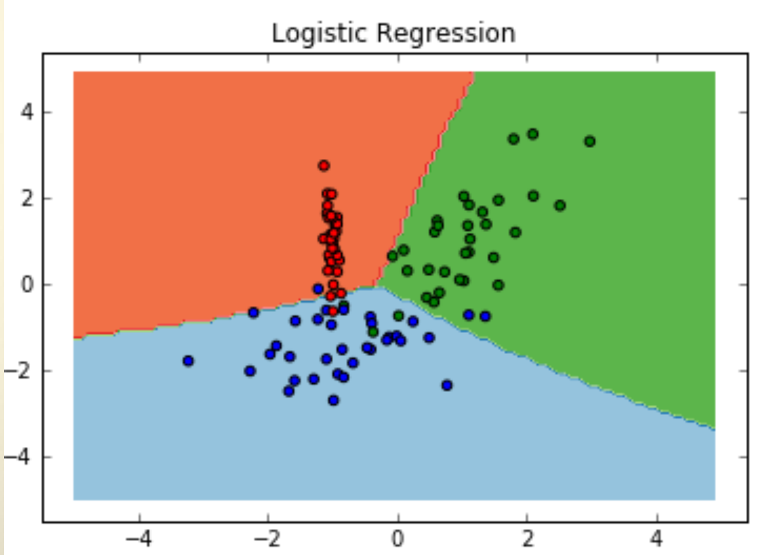




Dataset 2

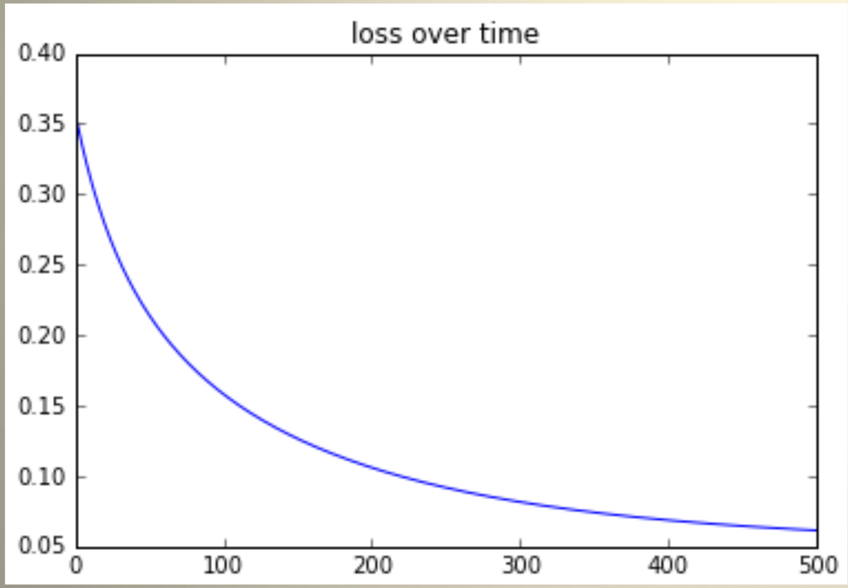
- Loss and Boundaries

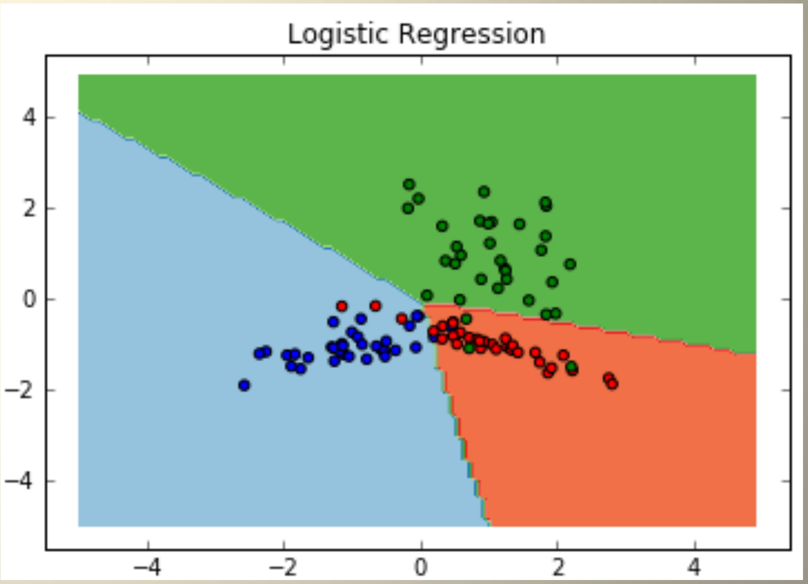




Dataset 3

- Loss and Boundaries





One-hot Matrix

- One-hot vector representation
  - For each training, its class label is represented as a K-dimensional binary vector, e.g. (0,1,0).

```
# #make data
X,Y = make_classification(n_features=2, n_informative=2, n_redundant=0,
                          n_repeated=0, n_classes=3, n_clusters_per_class=1)

#make Y into a one-hot matrix
lb = LabelBinarizer()
Y = lb.fit_transform(Y)
```

Plot Boundaries Function

```
#nice plotting code from before
def plot_contour_scatter(X, Y, model, title_text, binarizer):
    #sample from a lattice (for the nice visualization)
    x1, x2 = meshgrid(arange(-5,5,0.1), arange(-5,5,0.1))
    Xnew = vstack((x1.ravel(), x2.ravel())).T

    Z = model.predict(Xnew).reshape((Xnew.shape[0],-1))
    Zc = binarizer.inverse_transform(Z)
    y = binarizer.inverse_transform(Y)

    #plot - contour plot and scatter superimposed
    contourf(arange(-5,5,0.1), arange(-5,5,0.1), Zc.reshape(x1.shape),
             cmap ='Paired',levels=arange(0,4,0.1))
    c_dict = {0:'b', 1:'g', 2:'r', 3:'y', 4:'m', 5:'k', 6:'c'}
    colorsToUse=  [c_dict[yi] for yi in y]
    scatter(X[:,0], X[:,1], c=colorsToUse)
    title(title_text)
    show()
```

Questions?