MultiNomial Regression
ICP-Project Pattern Recognition

Neda Keivandarian

Abstract

Multinomial regression is used to predict categorical placement in or the probability of category membership on a dependent variable based on multiple independent variables. The independent variables can be either binary or continuous (i.e., interval or ratio in scale). Multinomial logistic regression is a simple extension of binary logistic regression that allows for more than two categories of the dependent. Like binary logistic regression, multinomial logistic regression uses maximum likelihood estimation to evaluate the probability of categorical membership. Here for this repository my attempt is to model MNR for iris data-set and find the decision boundaries for this model. I used softmax activation function, find the cross entropy and average cross entropy. And finally I use backtracking method to find the best η_max.

1 How MNR works

The multinomial logistic regression algorithm is an extension to the logistic regression model that involves changing the loss function to cross-entropy loss and predict probability distribution to a multinomial probability distribution to support multi-class classification problems.

Logistic regression is a classification algorithm. It is intended for datasets that have numerical input variables and a categorical target variable that has two values or classes. Problems of this type are referred to as binary classification problems.[1] Logistic regression is designed for two-class problems, modeling the target using a binomial probability distribution function. The class labels are mapped to 1 for the positive class or outcome and 0 for the negative class or outcome. The fit model predicts the probability that an example belongs to class 1.By default, logistic regression cannot be used for classification tasks that have more than two class labels, so-called multi-class classification.Instead, it requires modification to support multi-class classification problems.
One popular approach for adapting logistic regression to multi-class classification problems is to split the multi-class classification problem into multiple binary classification problems and fit a standard logistic regression model on each sub-problem. Techniques of this type include one-vs-rest and one-vs-one wrapper models.An alternate approach involves changing the logistic regression model to support the prediction of multiple class labels directly. Specifically, to predict the probability that an input example belongs to each known class label.The probability distribution that defines multi-class probabilities is called a multinomial probability distribution. A logistic regression model that is adapted to learn and predict a multinomial probability distribution is referred to as Multinomial Logistic Regression. Similarly, we might refer to default or standard logistic regression as Binomial Logistic Regression.

In this implementation softmax function is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. The softmax function takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval [0,1], and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.
softmax function model:

- A weight vectors for each class. The weight vectors are typically stored as rows in weight matrix

- A bias for each class
- Activation function which known as softmax function
- The cross-entropy loss function

The training procedure of a softmax regression model has different steps. In the beginning the model parameters are initialized. The other steps are repeated for a specified number of training iterations or until the parameters have converged.

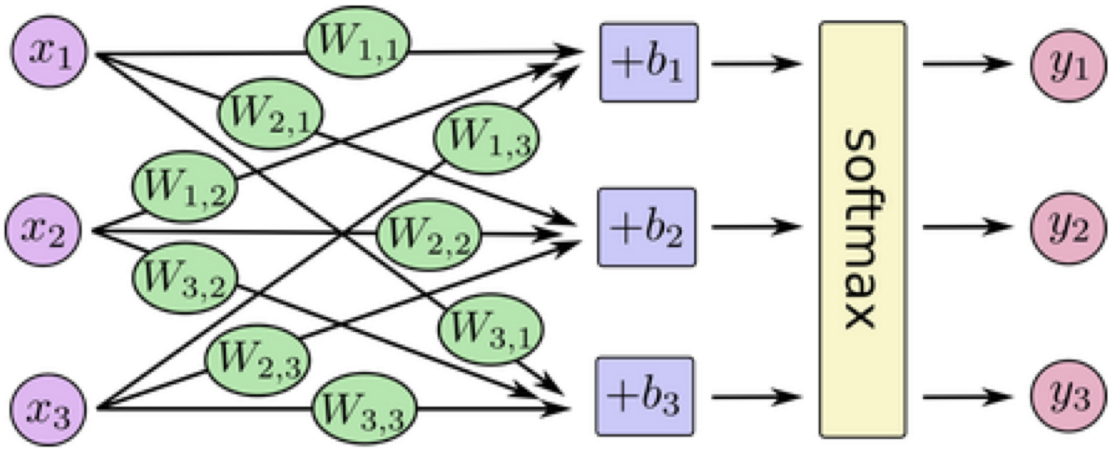

Figure 1: Whole procedure of MNR

1.1 Steps for creating MNR model

- **Step-1:** Initialize the weight matrix and bias values with zero or a small random values.
- **Step-2:** For each class of K we compute a linear combination of the input features and the weight vector of class K, which is for each training example compute a score for each class. For class K and input vector $\boldsymbol{x}_{(i)}$ we have:

  $s_k(\boldsymbol{x}_{(i)}) = \boldsymbol{w}_k^T \cdot \boldsymbol{x}_{(i)} + b_k$

  In this equation $\cdot$ is the dot product and $\boldsymbol{w}_{(k)}$ the weight vector of class K. So we can find and compute the s which is scores for all classes and training examples in parallel, using vectorization and broadcasting:

  $\boldsymbol{S} = \boldsymbol{X} \cdot \boldsymbol{W}^T + \boldsymbol{b}$

  which $\boldsymbol{X}$ is a matrix of data that has $n_{samples}$ and $n_{features}$ that holds all training examples, and $\boldsymbol{W}$ is a matrix in shape of $n_{classes}$ and $n_{features}$ that holds the weight vector for each class.

- **Step-3:** Apply the softmax function for activation to transform the scores into probabilities. The probability that an input vector $\boldsymbol{x}_{(i)}$ belongs to class k is given by:

  $\hat{p}_k(\boldsymbol{x}_{(i)}) = \frac{\exp(s_k(\boldsymbol{x}_{(i)}))}{\sum_{j=1}^{K} \exp(s_j(\boldsymbol{x}_{(i)}))}$

  In this step we have to perform all the formulas for all classes and our training examples when we using vectorization. We can see the class predicted by this model for $\boldsymbol{x}_{(i)}$ is then simply the class with the highest probability.

- **Step-4:** In this step we should calculate the cost over the whole training set. The result that we expected from this step is our model predict a high probability for our targeted class and the lowest probability for other classes. So it can use the cross entropy loss function:

  $h(\boldsymbol{W}, b) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log(\hat{p}_k^{(i)})\right]$

  Here we use one-hot encoding, because if we used numerical categories 1,2,3,... we would impute ordinal. So $y_k^{(i)}$ is 1 for the targeted class and for the other $\boldsymbol{x}^{(i)}$ for k classes $y_k^{(i)}$ should be 0.

- **Step-5:** In this step we need to compute gradient of the cost function for each weight vector and bias: for class k we have:

  $\nabla_{\boldsymbol{w}_k} h(\boldsymbol{W}, b) = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_{(i)}\left[\hat{p}_k^{(i)} - y_k^{(i)}\right]$

- **Step-6:** Here we just need to update biases and weights for all the classes of k an η is my learning rate or step length:

  $\boldsymbol{w}_k = \boldsymbol{w}_k - \eta\,\nabla_{\boldsymbol{w}_k} h$
  $b_k = b_k - \eta\,\nabla_{b_k} h$

2 Experimental Results

2.1 Decision Region for training MNR with No Regularization

In this project I use Iris dataset:

Figure 2: In this data-set we have 3 categories of iris flowers

For implementing the multinumial regression, I follow the above steps and train MNR with $\lambda = 0$ without regularization and show the obtained decision regions in a figure below.
To implement gradient descent with backtracking, the maximum learning rate is 300 and stopping the condition when $\frac{d_{ACE}}{d_w} < 0.01$. In this dataset, MNR uses linear decision boundary. Because linear boundaries are good enough to give very good rate.

- Training error for 120 samples is 0.02

decision regions for 2 features of petal length and petal width of iris flowers

In [54]:

```
#load the dataset
X,y = datasets.load_iris(return_X_y=True)
MNR = MultinumialRegression(thres=1e-5)
MNR.fit_model(X,y,lr=0.0001)
print(MNR.score(X, y))
fig = plt.figure(figsize=(8,6))
plt.plot(np.arange(len(MNR.loss)),MNR.loss)
plt.title(" Loss function(cross entropy) during training")
plt.xlabel("Number of iterations")
plt.ylabel("Loss")
plt.show()
```

```
0.98
```

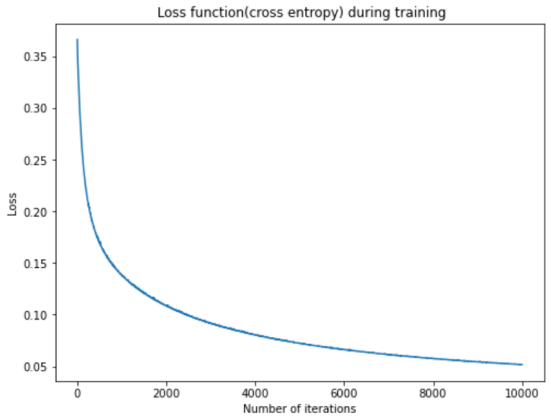

Figure 3: This figure represent monotonically decreasing average cross entropy versus number of iterations. This figure can prove that ACE function is minimized by training MNR.

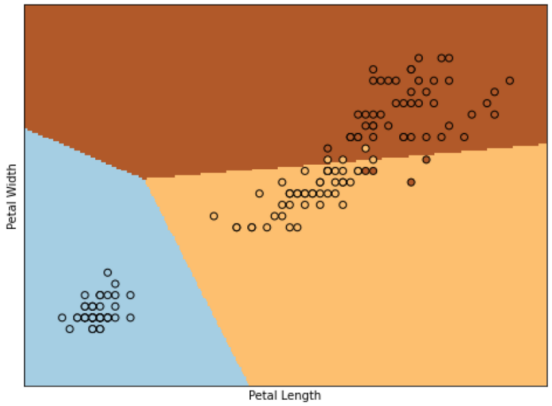

Figure 4: Decision region for training MNR with no regularization

2.2 Regularized MNR

By using non-linear transformation, we can use MNR with transformed data and by picking values of $\lambda = 10^k$, where k = -7:0.2:7, decision boundary is not linear anymore. In this method, we have a penalty term and try to make parameters small. So, the bigger penalty term is, parameters will be smaller. The effect of this technique is that the output of softmax is going to be smoother. In the limit, it is going to become constant and in this case, we predict samples based on the majority of labels not based on where samples lie on the feature space. In our example since we use the same portion of each class we will get constant error of 0.66 Misclassification rate of the resulting model on the hold-out set represented as following, the model exhibiting the least error as the champion MNR model was gained for λ between 0 to 0.631. In these cases, the least error is equal to zero.

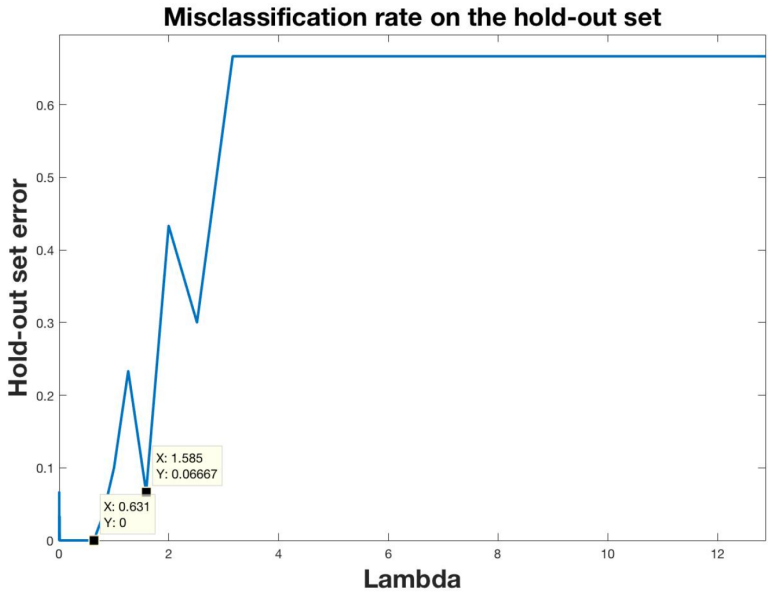

Figure 5: Misclassification rate on the hold-out set

I chose the second best MNR model's decision regions based on $\lambda = 1.585$, in this case hold-out set error is 0.06667 and training misclassification error is 0.125 . As you can see 40 samples form class 1 are classified as class 1, 15 samples of class 2 misclassified as class 3 and all samples from class 3 are classified correctly. So, the majority of the samples are classified as class 3.

Figure 6 depicts misclassification rate for training set. As we expect this rate is going up as the value of l increases, because small values for all parameters correspond to simpler model and less prone to overfitting. The maximum rate would be 0.6667, in comparison with MNR with no regularization in which this rate was 0.05.

We can also see in figure 7 that average cross entropy function is monotonically decreasing by non-linear transformation.

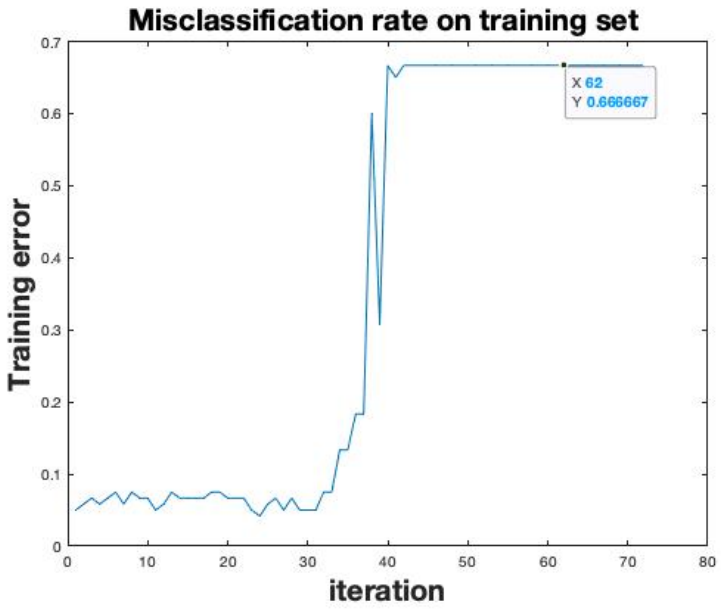

Figure 6: Misclassification rate on training set

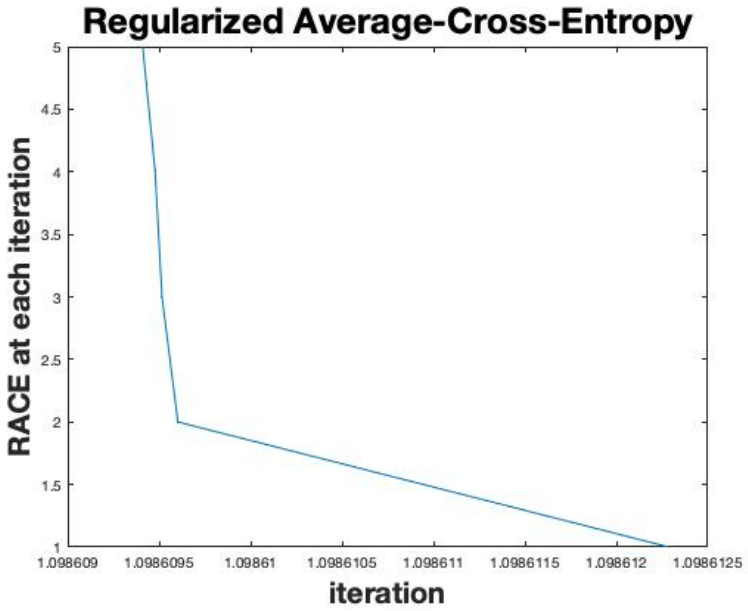

Figure 7: Regularized ACE

2.3 Gradient Descent Vs Stochastic Gradient Descent

- **Gradient Descent:** The gradient descent involves using the entire dataset or training set to compute the gradient to find the optimal solution. Our movement towards the optimal solution, which could be the local or global optimal solution, is always direct. Using this variant of the model to update for a parameter in a particular iteration requires that we run through all the samples of our training set every time we want to create a single update[2]
  However, this can become a major challenge when we have to run through millions of samples. And because a gradient descent example involves running through the entire data set during each iteration, we will spend a lot of time and computational strength when we have millions of samples to deal with. Not only is this difficult, but it is also very unproductive.

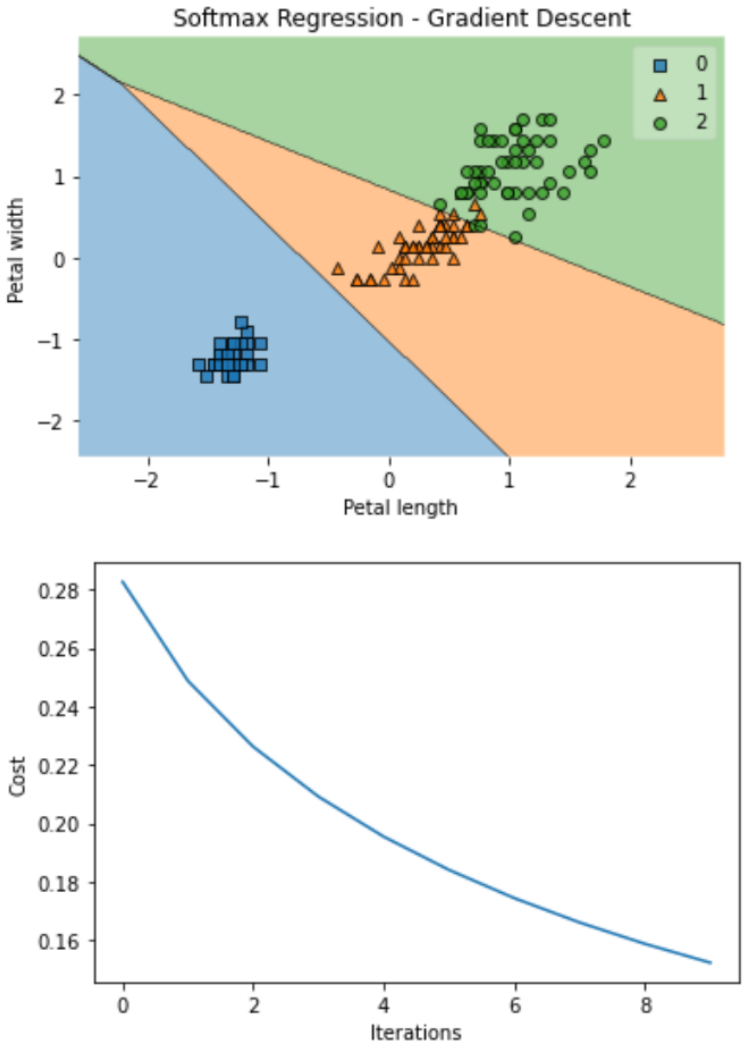

Figure 8: SoftmaxRegression-Gradient Descent

- **Stochastic Gradient Descent:** SGD is a variant of the optimization algorithm that saves us both time and computing space while still looking for the best optimal solution. In SGD, the dataset is properly shuffled to avoid pre-existing orders then partitioned into m examples.[3] This way the stochastic gradient descent python algorithm can then randomly pick each example of the dataset per iteration (as opposed to going through the entire dataset at once). A stochastic gradient descent example will only use one example of the training set for each iteration. And by doing so, this random approximation of the data set removes the computational burden associated with gradient descent while achieving iteration faster and at a lower convergence rate. The process simply takes one random stochastic gradient descent example, iterates, then improves before moving to the next random example.

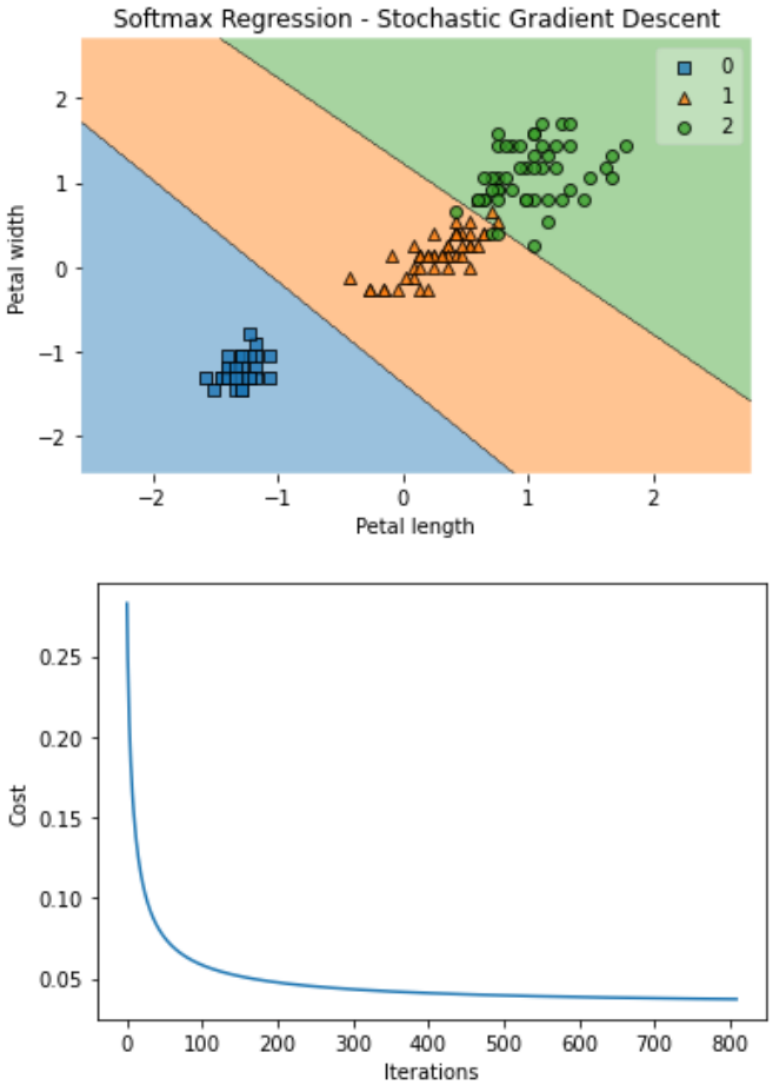

Figure 9: SoftmaxRegression-Stochastic Gradient Descent

References

[1] Juliana Tolles and William J. Meurer. Logistic Regression: Relating Patient Characteristics to Outcomes. *JAMA*, 316(5):533–534, 08 2016.

[2] S. Sra, S. Nowozin, and S.J. Wright. *Optimization for Machine Learning*. Neural information processing series. MIT Press, 2012.

[3] Philip Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.