Logistic regression via gradient descent (master)

Out[2]: Click here to display/hide the code.

The random seed is set so results are repeatable.

Gradients

The code used here to compute the gradient of a function is different from the code used in the linear regression assignment. This code is more efficient. Please read the code carefully.

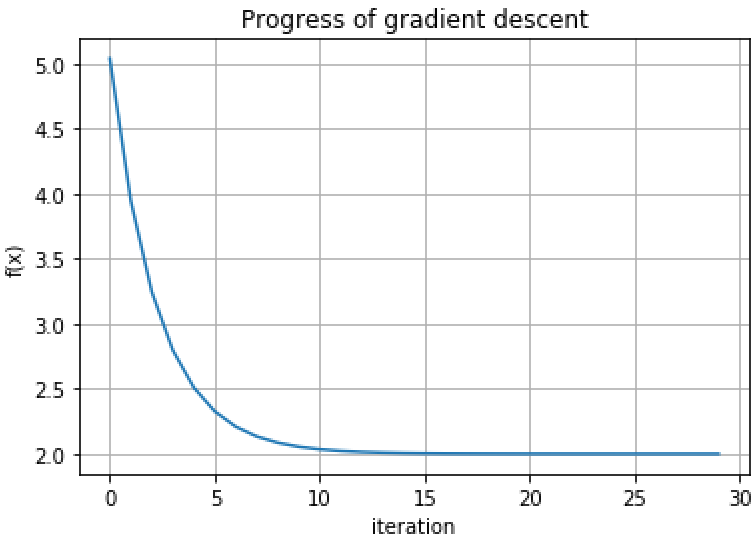
```
x: [1. 2.], f_grad(x): [6.000002 1.      ]
x: [0. 0.], f_grad(x): [2.e-06 0.e+00]
x: [ 2. -1.], f_grad(x): [7.000002 2.      ]
```

Gradient descent

In gradient descent we want to find the value of x that minimizes (or maximizes) a function f. We do this by starting with some x, computing the value of the gradient of f at x, and then using that value to make an adjustment to x.

```
x = [0.99667033], f(x) = 0.000
x = [-1.10462872], f(x) = -0.649
x = [ 0.99923113 -0.99701226], f(x) = 2.000
```

Plot the value of the loss function as gradient descent proceeds.



Binary Logistic regression

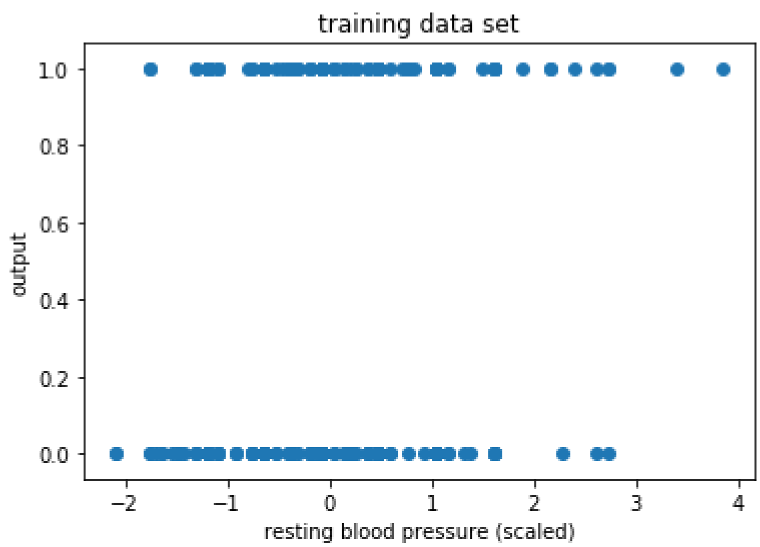
The key idea for training is that we want to use gradient descent to find the model parameters that minimize the loss function. For binary classification with logistic regression, the loss function is "log loss".

Heart disease data set

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   age        270 non-null    int64  
 1   sex        270 non-null    int64  
 2   chestpain  270 non-null    int64  
 3   restbp     270 non-null    int64  
 4   chol       270 non-null    int64  
 5   sugar      270 non-null    int64  
 6   ecg        270 non-null    int64  
 7   maxhr      270 non-null    int64  
 8   angina     270 non-null    int64  
 9   dep        270 non-null    float64
 10  exercise   270 non-null    int64  
 11  fluor      270 non-null    int64  
 12  thal       270 non-null    int64  
 13  output     270 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 29.7 KB
```

Prepare the training data set.

```
(270, 4)
(270,)
```



Log loss

Note that to compute the loss we need both the model parameters and the training data.

```
0.2689414213699951
0.5
0.6899744811276125
```

```
b: [-0.2  0.1  0.3 -1. ], log_loss(b): 0.582
b: [0.8 0.5 0.6 0.9], log_loss(b): 1.021
b: [0.05 0.08 0.02 0.8 ], log_loss(b): 0.923
```

Gradient descent with log loss

Now we can put together our gradient descent function and log_loss function to perform logistic regression.

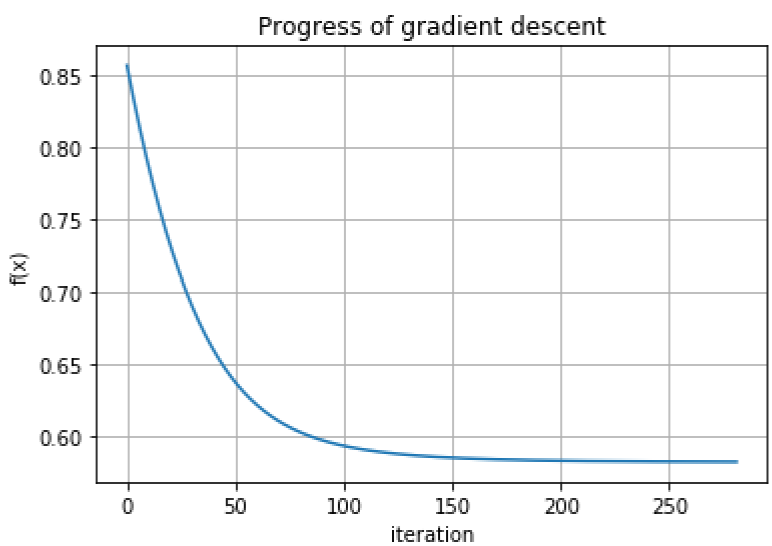
We create a loss function that is log_loss() with our training data X1, y "hardwired" in.

```
[-0.24640406  0.07479957  0.30294116 -0.93332161]
```

Compute training loss and training accuracy. Use a threshold of 0.5 when computing accuracy.

```
training loss: 0.582, training accuracy: 0.711
```

This plot shows how log loss decreases during gradient descent.



Compare to the result from Scikit-Learn

```
Model coefficients:
[-0.2463097]
[[ 0.04820012  0.31247721 -0.94939172]]

Scikit-Learn training accuracy: 0.704
```

Multi-class logistic regression

If we have a multi-class classification problem, then we need to use a separate linear model for

If we have a multi-class classification problem, then we need to use a separate linear model for each output class. If we have 3 output class, for example, from each input we will get a vector of three output values. In multi-class logistic regression, we transform this vector to a vector of probabilities using the softmax function.

Iris data set

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Out[22]:
```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

Preprocess the data

```
(150, 5)
(150, 3)
(150,)
```

Softmax

Test softmax

```
[0.30166396 0.60747661 0.09085944]
[0.16232789 0.65827205 0.17940006]
```

The generalization of log loss to more than 2 classes is cross entropy.

Making predictions

We have a separate linear model for each output class. If we have k output classes, and n predictors, then we need n+1 parameters for each class. Instead of keeping the parameters for each class separate, we can put them in a single matrix B. Each row of B will contain the parameters for one class. In other words, B will have k rows and n+1 columns.

Think about the matrix multiplication that is needed here. Previously we got a single output value for each training example by computing X1.dot(b). How shall we combine X1 and B to get an array of outputs, one for each class?

Make predictions using a random set of parameters of the linear model. Each row in array pred is a prediction.

Out[26]:
```
array([[-1.79440354, -1.02495559,  0.05587703],
       [-2.04197974, -2.00050214, -0.86088335],
       [-2.30642572, -1.69000608, -0.56916167],
       [-2.324334  , -1.77903546, -0.66800388],
       [-1.90190191, -0.84189648,  0.22444893]])
```

Apply softmax to each row, so that the values in each row sum to 1.

```
[[0.10504005 0.22673675 0.66822321]
 [0.18866878 0.19665887 0.61467236]
 [0.1171775  0.21704605 0.66577645]
 [0.12554662 0.21658341 0.65786997]
 [0.08149559 0.23522779 0.68327662]]
```

Cross-entropy

Test cross_entropy().

```
0.020202707317519466
0.10536051565782628
0.916290731874155
1.6094379124341003
0.4942963218147801
1.2039728043259361
```

```
1.0121847560247488
```

Test cross_entropy_loss().

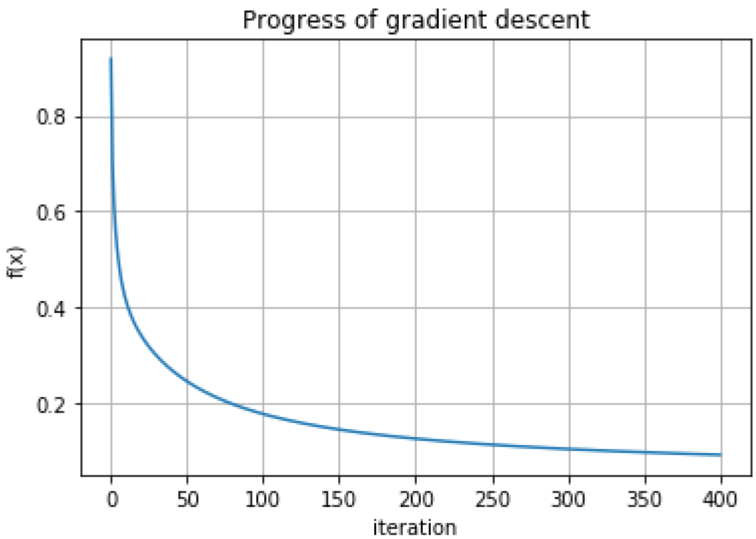
```
0.9810094431655981
0.7759886209662975
```

Gradient descent with cross-entropy.

Using gradient descent, softmax, and cross entropy loss we can perform multi-class logistic regression.

```
warning: reached iteration limit
[ 0.07220324  3.3688566  -1.71313402 -0.98973188  1.27655259  0.91423354
  2.12894222  0.16243843 -0.49257505 -2.39038444  0.1098489   3.52904675
 -1.62851411 -0.46070803  4.33171562]
```

```
0.09041396997208528
```



Compute training set accuracy

```
gradient descent training accuracy: 0.973
```

Compare results to Scikit Learn

```
Scikit-Learn training accuracy: 0.973
```