PyTorch Logistic Classifier Example

November 7, 2018

1 PyTorch Logistic Classifier Example

Please email me (tdm47@case.edu) if you have questions.

Packages required: PyTorch, pandas, matplotlib,

Import the dataset as a pandas dataframe. Remove the last iris class.

In [1]:
```
import pandas as pd
dataframe = pd.read_csv('./irisdata.csv')

two_class = dataframe[dataframe['species'] != 'setosa']
```

Encode the class labels as 0 and 1. This is required for the logistic classifier.

In [2]:
```
# The warnings can be ignored
two_class.loc[two_class['species'] == 'virginica', 'species'] = 0
two_class.loc[two_class['species'] == 'versicolor', 'species'] = 1
```

```
/usr/local/lib/python2.7/site-packages/pandas/core/indexing.py:543: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self.obj[item] = s
```

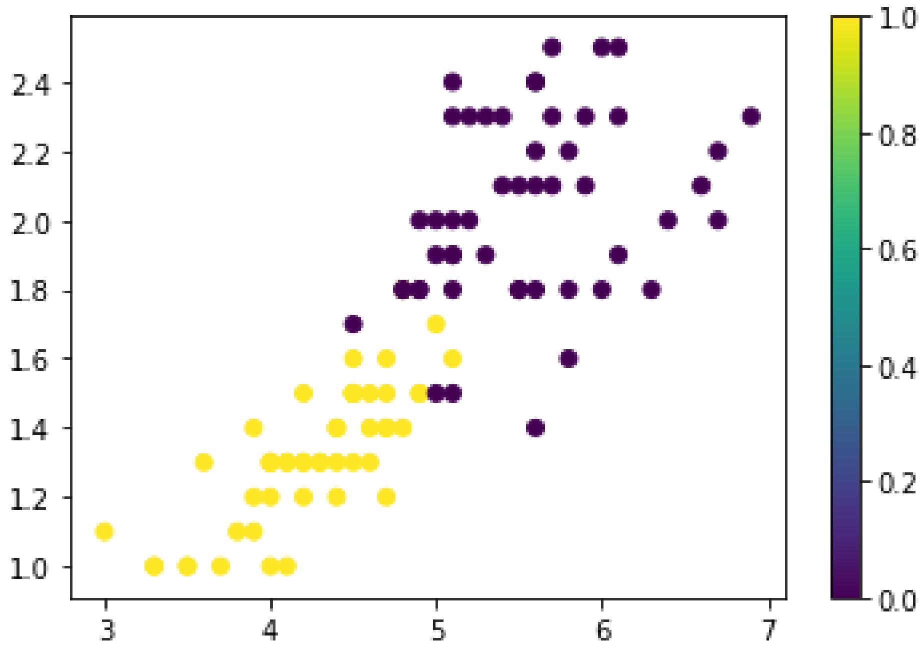
Specify the input attribute vectors and output vectors for the network.

In [3]:
```
in_vec = two_class[['petal_length', 'petal_width']]
out_vec = two_class['species']
```

1.0.1 Plot of Classes

To help visualize the problem, the verginica and versicolor classes have been plotted. Virginica [0] is denoted by the purple dots, and versicolor [1] is denoted by the yellow dots.

In [9]:
```
import matplotlib.pyplot as plt
plt.scatter(in_vec.values[:,0], in_vec.values[:,1], c=out_vec.values)
plt.colorbar()
plt.show()
```



Import `torch`, and define the network structure. The `forward` method defines the flow of information through the model. In this single layer example, an input to the network is first multiplied by the weights `self.fullyconnected1(x)`, then sent through the activation function `F.sigmoid(x)`.

`nn.Linear` is an abstraction of a layer for a feed forward network. It contains the weights of that layer. An input sent to the the instantiated object multiplies the input by the weights.

```
In [5]: import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim

        num_in = 2 # size of input attributes
        num_out = 1 # size of output

        class Network(nn.Module):
            def __init__(self):
                super(Network, self).__init__()
                self.fullyconnected1 = nn.Linear(num_in,num_out)

            def forward(self, x):
                x = self.fullyconnected1(x)
                x = F.sigmoid(x)
                return x
```

Instantiate the network, loss function, and optimizer.

```
In [6]: model = Network()

        criterion = nn.MSELoss() # loss function
        optimizer = torch.optim.SGD(model.parameters(), lr=0.01) # try tuning the learning rat
```

1.0.2 Train the Classifier

```
In [7]: num_epochs = 1000 # number of training iterations
        num_examples = two_class.shape[0]

        model.train()

        for epoch in range(num_epochs):
            for idx in range(num_examples):

                # for example `idx`, convert data to tensors so that PyTorch can use it.
                attributes = torch.tensor(in_vec.iloc[idx].values, dtype=torch.float)
                label = torch.tensor(out_vec.iloc[idx], dtype=torch.float)

                # reset the optimizer's gradients
                optimizer.zero_grad()

                # send example `idx` through the model
                output = model(attributes)

                # compute gradients based on error
                loss = criterion(output, label)
                # propegate error through network
                loss.backward()

                # update weights based on propegated error
                optimizer.step()

            if(epoch % 100 == 0):
                print('Epoch: {} | Loss: {:.6f}'.format(epoch, loss.item()))
```

```
Epoch: 0 | Loss: 0.038957
Epoch: 100 | Loss: 0.045446
Epoch: 200 | Loss: 0.047594
Epoch: 300 | Loss: 0.048659
Epoch: 400 | Loss: 0.049173
Epoch: 500 | Loss: 0.049345
Epoch: 600 | Loss: 0.049287
Epoch: 700 | Loss: 0.049073
Epoch: 800 | Loss: 0.048751
Epoch: 900 | Loss: 0.048355
```

1.0.3 Test the Classifier

```
In [8]: model.eval()

        pred = torch.zeros(out_vec.shape)

        for idx in range(num_examples):

            attributes = torch.tensor(in_vec.iloc[idx].values, dtype=torch.float)
            label = torch.tensor(out_vec.iloc[idx], dtype=torch.float)

            # save the predicted value
            pred[idx] = model(attributes).round()

        print('Correct classifications: {}/{}'.format(sum(pred == torch.tensor(out_vec.values)
```

```
Correct classifications: 93/100
```