Perceptron Models

```
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
from sklearn.datasets import make_classification
X, y = make_classification(20, 2,2,0, weights=[.5,.5], random_state=13579)
clf = Perceptron(max_iter=250, tol=1e-3, fit_intercept= True, eta0= 0.01, random_state=12210)
clf.fit(X, y)

print("Accuracy Score of Perceptron Model: %.3f" % clf.score(X, y))
```

```
Accuracy Score of Perceptron Model: 0.750
```

```
# Create a perceptron object with the parameters: 40 iterations (epochs) over the data, and a
print('X1 coefficient:' + str(clf.coef_[0,0]))
print('X2 coefficient:' + str(clf.coef_[0,1]))
print('Intercept:' + str(clf.intercept_))
```

```
X1 coefficient:0.02170702951150036
X2 coefficient:-0.017528374768389636
Intercept:[0.]
```

Application of the **Perceptron Model** to the Iris Classification

```
# Load required libraries
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
# Load the iris dataset
iris = datasets.load_iris()

# Create our X and y data
X = X = iris.data[:, [0, 2]]
y = iris.target
# View the first five observations of our y data
y[:5]
```

```
array([0, 0, 0, 0, 0])
```

```
# View the first five observations of our x data.
# Notice that there are four independent variables (features)
X[:5]
```

```
array([[5.1, 1.4],
       [4.9, 1.4],
       [4.7, 1.3],
       [4.6, 1.5],
       [5. , 1.4]])
```

```
# Split the data into 70% training data and 30% test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
# Train the scaler, which standarizes all the features to have mean=0 and unit variance
sc = StandardScaler()
sc.fit(X_train)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
# Apply the scaler to the X training data
X_train_std = sc.transform(X_train)

# Apply the SAME scaler to the X test data
X_test_std = sc.transform(X_test)
```

```
# Create a perceptron object with the parameters: 40 iterations (epochs) over the data, and a
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=0)

# Train the perceptron
ppn.fit(X_train_std, y_train)
```

```
Perceptron(alpha=0.0001, class_weight=None, early_stopping=False, eta0=0.1,
           fit_intercept=True, max_iter=40, n_iter_no_change=5, n_jobs=None,
           penalty=None, random_state=0, shuffle=True, tol=0.001,
           validation_fraction=0.1, verbose=0, warm_start=False)
```

```
# Apply the trained perceptron on the X data to make predicts for the y test data
y_pred = ppn.predict(X_test_std)
```

```
# View the predicted y test data
y_pred
```

```
array([2, 2, 2, 0, 2, 1, 1, 1, 0, 2, 0, 2, 1, 1, 0, 2, 1, 1, 1, 1, 2, 1,
       1, 1, 0, 1, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 1, 0, 1, 1, 1, 2, 0,
       1])
```

```
# View the true y test data
y_test
```

```
array([2, 2, 2, 0, 2, 1, 1, 1, 0, 2, 0, 2, 1, 1, 0, 2, 1, 1, 2, 1, 2, 2,
       2, 1, 0, 1, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 1, 1, 1, 2, 0,
       2])
```

```
# View the accuracy of the model, which is: 1 - (observations predicted wrong / total observa
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('Actuals: ' + str(y))
print('Predictions: ' + str(ppn.predict(X)))
```

```
Accuracy: 0.889
Actuals: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Predictions: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
print('X1 coefficient:' + str(ppn.coef_[0,0]))
print('X2 coefficient:' + str(ppn.coef_[0,1]))
print('Intercept:' + str(ppn.intercept_))
```

```
X1 coefficient:0.03019456359962991
X2 coefficient:-0.3210844590578492
Intercept:[-0.2 -0.1 -0.5]
```

```
X = iris.data[:, [0, 2]]
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
# Plotting decision regions
plot_decision_regions(X, y, clf=ppn, legend=2)

# Adding axes annotations
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.title('SVM on Iris')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244: Matplot
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```

Perceptron Models with PyTorch

```
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(1,1)
    def forward(self, x):
        x = self.fc1(x)
        return x


net = Net()
print(net)
```

```
Net(
  (fc1): Linear(in_features=1, out_features=1, bias=True)
)
```

```
print(list(net.parameters()))
```

```
[Parameter containing:
tensor([[0.4753]], requires_grad=True), Parameter containing:
tensor([-0.2086], requires_grad=True)]
```

```
input = Variable(torch.randn(1,1,1), requires_grad=True)
print(input)
```

```
tensor([[[0.5059]]], requires_grad=True)
```

```
out = net(input)
print(out)
```

```
tensor([[[0.0319]]], grad_fn=<AddBackward0>)
```

```
import torch.optim as optim
def criterion(out, label):
    return (label - out)**2
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)
```

```
data = [(1,3), (2,6), (3,9), (4,12), (5,15), (6,18)]


for epoch in range(100):
    for i, data2 in enumerate(data):
        X, Y = iter(data2)
        X, Y = Variable(torch.FloatTensor([X]), requires_grad=True), Variable(torch.FloatTens
        optimizer.zero_grad()
        outputs = net(X)
        loss = criterion(outputs, Y)
        loss.backward()
        optimizer.step()
        if (i % 10 == 0):
            print("Epoch {} - loss: {}".format(epoch, loss.data[0]))
```

```
Epoch 0 - loss: 7.470733165740967
Epoch 1 - loss: 1.0347850322723389
Epoch 2 - loss: 0.07864704728126526
Epoch 3 - loss: 0.233488529920578
Epoch 4 - loss: 0.1468745321035385
Epoch 5 - loss: 0.15211068093776703
Epoch 6 - loss: 0.1301565319299698
Epoch 7 - loss: 0.11941292136907578
Epoch 8 - loss: 0.10674891620874405
Epoch 9 - loss: 0.09634831547737122
Epoch 10 - loss: 0.08665211498737335
Epoch 11 - loss: 0.07803431153297424
Epoch 12 - loss: 0.0702395886182785
Epoch 13 - loss: 0.06323456764221191
Epoch 14 - loss: 0.05692450702190399
Epoch 15 - loss: 0.051245469599962234
Epoch 16 - loss: 0.04613243415951729
Epoch 17 - loss: 0.041529856622219086
Epoch 18 - loss: 0.037386465817689896
Epoch 19 - loss: 0.03365617245435715
Epoch 20 - loss: 0.03029816597700119
Epoch 21 - loss: 0.027275364845991135
Epoch 22 - loss: 0.02455403469502926
Epoch 23 - loss: 0.0221041738986969
Epoch 24 - loss: 0.019898829981684685
Epoch 25 - loss: 0.017913423478603363
Epoch 26 - loss: 0.01612623780965805
Epoch 27 - loss: 0.0145172830671072
Epoch 28 - loss: 0.013068907894194126
Epoch 29 - loss: 0.01176500879228115
Epoch 30 - loss: 0.010591212660074234
Epoch 31 - loss: 0.009534461423754692
Epoch 32 - loss: 0.00858321599662304
Epoch 33 - loss: 0.007726815529167652
Epoch 34 - loss: 0.006955919787287712
Epoch 35 - loss: 0.006261848378926516
Epoch 36 - loss: 0.005637173075228929 5
Epoch 37 - loss: 0.005074654705822468
Epoch 38 - loss: 0.004568407777696848
```

```
Epoch 39 - loss: 0.004112573806196451
Epoch 40 - loss: 0.00370227568782866
Epoch 41 - loss: 0.0033328859135508537
Epoch 42 - loss: 0.00300040515139699
Epoch 43 - loss: 0.002701005432754755
Epoch 44 - loss: 0.002431520028039813
Epoch 45 - loss: 0.002188891638070345
Epoch 46 - loss: 0.0019705535378307104
Epoch 47 - loss: 0.0017739119939506054
Epoch 48 - loss: 0.0015969466185197234
Epoch 49 - loss: 0.0014376008184626698
Epoch 50 - loss: 0.0012941820314154029
Epoch 51 - loss: 0.0011650588130578399
Epoch 52 - loss: 0.0010487954132258892
Epoch 53 - loss: 0.0009441576548852026
Epoch 54 - loss: 0.000849973235744983
Epoch 55 - loss: 0.0007651488413102925
Epoch 56 - loss: 0.0006888186908327043
Epoch 57 - loss: 0.0006201148498803377
Epoch 58 - loss: 0.0005582146113738418
```

```
print(list(net.parameters()))
```

```
[Parameter containing:
tensor([[2.9994]], requires_grad=True), Parameter containing:
tensor([0.0032], requires_grad=True)]
```

```
print(net(Variable(torch.Tensor([[[1]]]))))
```

```
tensor([[[3.0026]]], grad_fn=<AddBackward0>)
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(1,10)
        self.fc2 = nn.Linear(10,1)
    def forward(self, x):
        x = self.fc2(self.fc1(x))
        return x
```

```
net = Net()
net.cuda()
```

```
Net(
  (fc1): Linear(in_features=1, out_features=10, bias=True)
  (fc2): Linear(in_features=10, out_features=1, bias=True)
)
```

```
X, Y = Variable(torch.FloatTensor([X]), requires_grad=True).cuda(), Variable(torch.FloatTenso
```

```
print(X)
```

```
tensor([6.], device='cuda:0', grad_fn=<CopyBackwards>)
```

```
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Perceptron with PyTorch

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(1,1)
        self.fc2 = nn.Linear(1,1)
    def forward(self, x):
        x = F.relu(self.fc2(F.relu(self.fc1(x))))
        return x
criterion = nn.MSELoss()
```

```
print(list(net.parameters()))
```

```
[Parameter containing:
tensor([[ 0.5287],
        [ 0.5206],
        [-0.6515],
        [-0.9662],
        [ 0.3464],
        [-0.1690],
        [ 0.1935],
        [-0.1135],
        [-0.7408],
        [-0.1673]], device='cuda:0', requires_grad=True), Parameter containing:
tensor([ 0.5648,  0.2877,  0.4944, -0.7134, -0.8644, -0.6519,  0.7540,  0.1194,
        -0.4214, -0.6301], device='cuda:0', requires_grad=True), Parameter containing:
tensor([[ 0.1109,  0.1716, -0.3134,  0.1576,  0.2911, -0.2719,  0.2034, -0.2665,
         -0.2725,  0.0492]], device='cuda:0', requires_grad=True), Parameter containing
tensor([0.2350], device='cuda:0', requires_grad=True)]
```

✓ 0s completed at 10:25 AM