Perceptron Classifier

Riley Kopp

20 September 2020

1 Create The Perceptron Class

1.1 How to Use the Perceptron:

To use this perceptron in your own code, use the following line

```
from ML import Perceptron
```

Then create the an object of the perceptron by running:

```
classifier = Perceptron(<scale>, <epochs>)
```

Then fit the model by calling:

```
classifier.fit(X, y)
```

Where X is an $m \times n$ matrix where m is the number of training data points and n is the number of features within each datapoint and where y is a vector of size m that contains a the class of the corresponding datapoint in X.

Once the model is fitted, we have a few more options. To view the weights used by the perceptron call `classifer.get_weights()`, this will return a vector of size w where $w = n + 1$ and w_0 is the bias.

To view the result of the dot product within the Perceptron, call the function: `classifier.net_input(X)`.

To run the full classifer on testing data, call: `classifer.predict(X)`

1.2 What does the Perceptron Do?

At a high level, the Perceptron is a binary classifier, meaning that it can separate items into two classes based on their features. At a lower level, the perceptron takes a dot product of the feature set X_i and the weight vector w then adds the bias. From there, the sum of the dot product and the bias, u, is run though the step function defined below.[1]

$$S(u) = \begin{cases} 1 & \text{if } u > 0 \\ -1 & \text{if } u \leq 0 \end{cases}$$

The weights are determined during `fit()` using the following equation:

$$w = w + ((y_{actual} - y_{predict}) * scale * X_i)$$

And the bias is determined using:

$$bias = bias + ((y_{actual} - y_{predict}) * scale)$$

Both w and $bias$ are updated after every prediction using the following algorithm:

Algorithm 1: Perceptron Fitting

Result: a trained perceptron
for *each epoch* **do**
  $error = 0$
  for *i in X* **do**
    predict = classifier.fit(i);
    $w = w + ((y_{actual} - y_{predict}) * scale * X_i)$
    $bias = bias + ((y_{actual} - y_{predict}) * scale)$
    $error = error + |y_{actual} - y_{predict}|$
  end
  if $error$ *is 0* **then**
    break;
  end
end

Once the perceptron is fit. The percptron can run on the test data.

[1] The perceptron source code is attached in Appendix A as to not bore the reader.

2 Results

For testing, the perceptron was trained on the first 100 datapoints in the Iris dataset [2]. In order to visualize the decision regions, we were provided the function `plot_decision_regions()` which gives us the following graphs after the classifier has been trained.

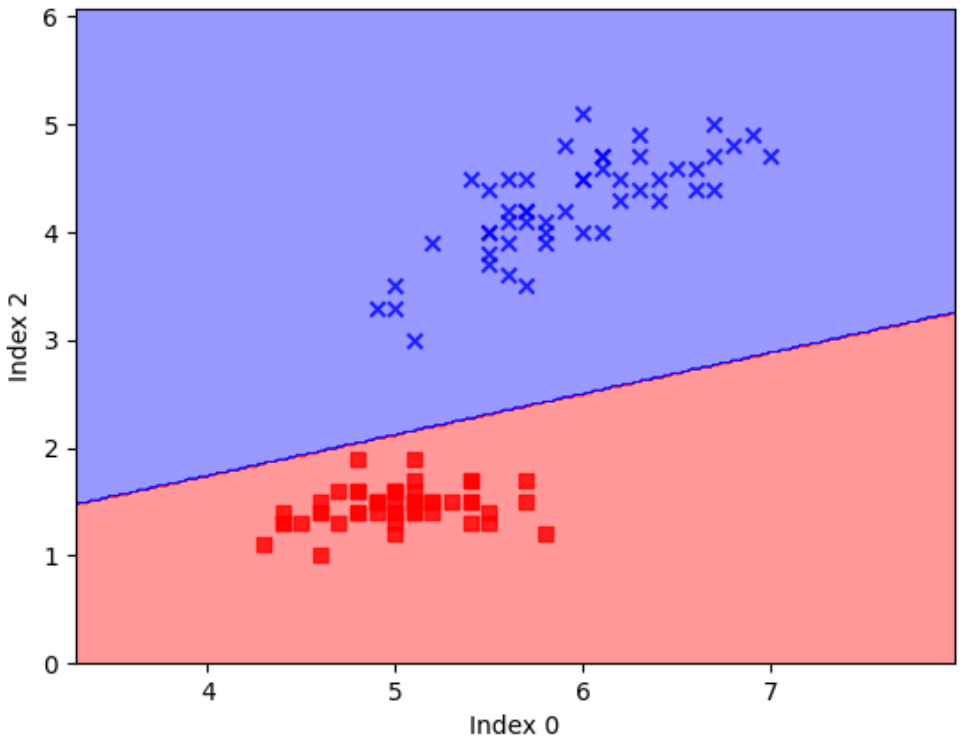

Figure 1: Example decision regions for index 0 and 2

[2] https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

2.1 2 Feature Classification

Once I was able to reproduce the example regions. I fit a model for each of the other combinations of features. The only model that "failed" was the model trained on indices 0 and 1, Since there is no linear equation that can separate the classes.

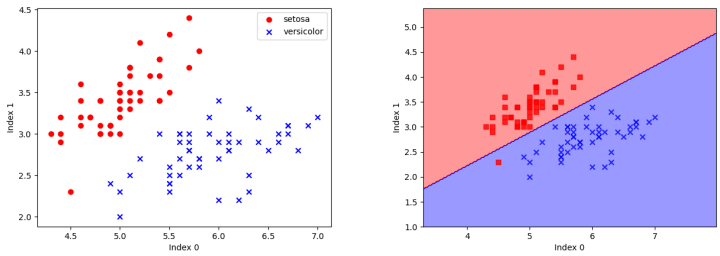

Figure 2: Decision regions for index 0 and 1 with some error

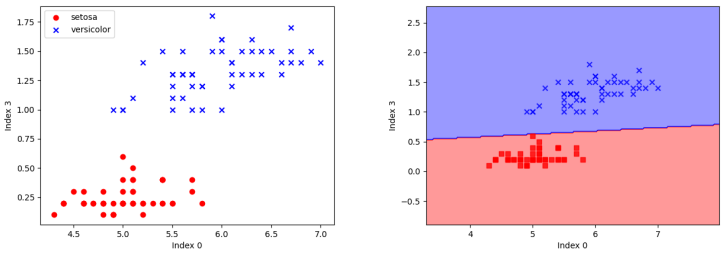

Figure 3: Decision regions for index 0 and 3

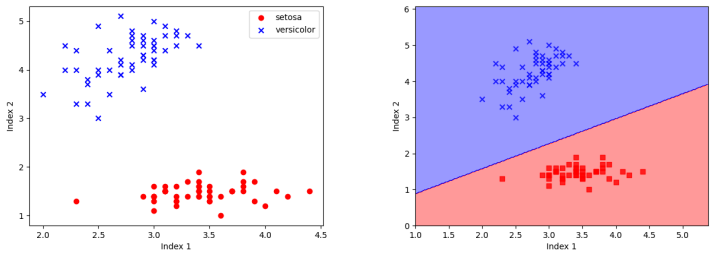

Figure 4: Decision regions for index 1 and 2

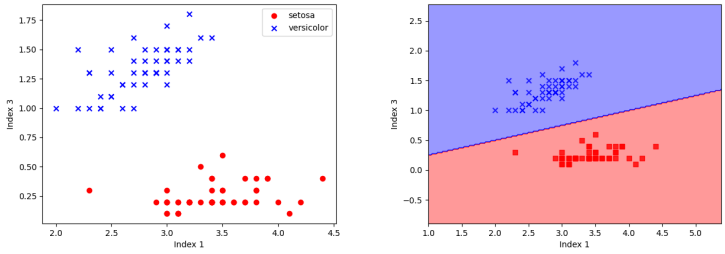

Figure 5: Decision regions for index 1 and 3

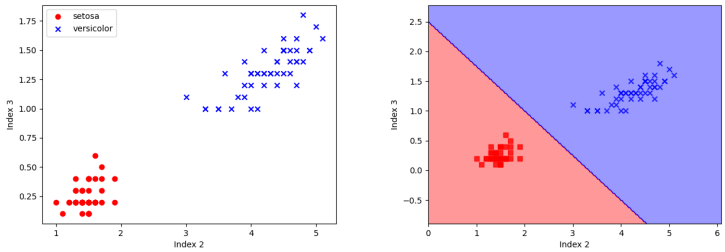

Figure 6: Decision regions for index 2 and 3

2.2 3 Feature Classification

I also played around with 3 Features and modified the `plot_decision_regions()`[3] to also plot in 3D and plots a plane to divide the regions in three space. (Clearly 3D is hard to represent on a 2D piece of paper, but I tried to get an angle on the graph that showed the separation of the clusters as best I could)

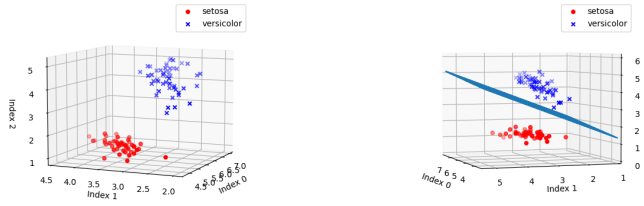

Figure 7: Decision regions for index 0, 1 and 2

[3] See Appendix A for the modifications.

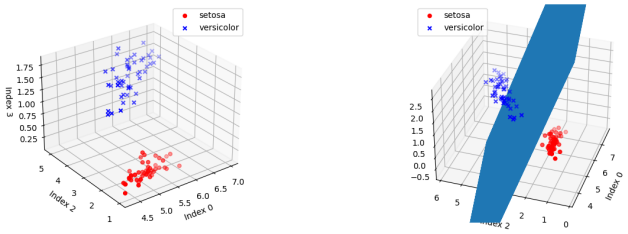

Figure 8: Decision regions for index 0, 2 and 3

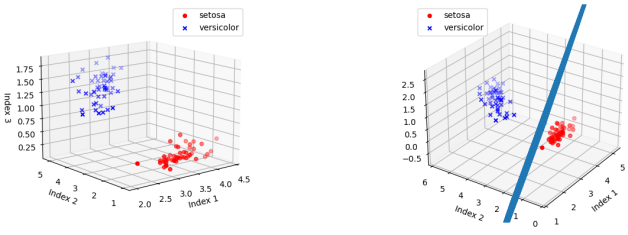

Figure 9: Decision regions for index 1, 2 and 3

A Code

A.1 Main

```
# This is a sample Python script.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from ML import Perceptron

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool
↪  windows, actions, and settings.
from matplotlib.colors import ListedColormap


def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    shape = X.shape
    if shape[1] == 2:
        markers = ('s', 'x', 'o', '^', 'v')
        colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
        cmap = ListedColormap(colors[:len(np.unique(y))])
        # plot the decision surface
        x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                               np.arange(x2_min, x2_max, resolution))
        Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
        Z = Z.reshape(xx1.shape)
        plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
        plt.xlim(xx1.min(), xx1.max())
        plt.ylim(xx2.min(), xx2.max())
        # plot class samples
        for idx, cl in enumerate(np.unique(y)):
            plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                        alpha=0.8, c=cmap(idx),
                        marker=markers[idx], label=cl)

    elif shape[1] == 3:
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
        cmap = ListedColormap(colors[:len(np.unique(y))])
        # plot the decision surface
        x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        x3_min, x3_max = X[:, 2].min() - 2, X[:, 2].max() + 1
```

```
        xx1, xx2, xx3 = np.meshgrid(np.arange(x1_min, x1_max,
        ↪  resolution),
                                    np.arange(x2_min, x2_max, resolution),
                                    np.arange(x3_min, x3_max, resolution))
        l1, l2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                                         np.arange(x2_min, x2_max,
                                         ↪  resolution))
        # Z = classifier.net_input(np.array([xx1.ravel(), xx2.ravel(), xx3.ravel()]).T)
        w = classifier.get_weights()
        Z = -1 * (w[0] + w[1] * l1.ravel() + w[2] * l2.ravel()) / w[3]
        Z.reshape(l1.shape)
        print(Z.shape)
        ax.plot(l1.ravel(), l2.ravel(), Z)
        ax.set_xlim(xx1.min(), xx1.max())
        ax.set_ylim(xx2.min(), xx2.max())
        ax.set_zlim(xx3.min(), xx3.max())
        ax.scatter(X[:50, 0], X[:50, 1], X[:50, 2], color='red',
        ↪  marker='o', label='setosa')
        ax.scatter(X[50:100, 0], X[50:100, 1], X[50:100, 2],
        ↪  color='blue', marker='x', label='versicolor')
        ax.legend()


def main():
    # Use a breakpoint in the code line below to debug your script.
    df =
    ↪  pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
    ↪  header=None)

    y = df.iloc[0:100, 4].values
    print(df.head(1))
    y = np.where(y == 'Iris-setosa', -1, 1)

    x = df.iloc[0:100, [0, 2]].values
    xlabel_text = "Index 0"
    ylabel_text = "Index 2"
    zlabel_text = "Index 3"
    #
    # fig = plt.figure()
    # ax = fig.add_subplot(111, projection = '3d')
    # ax.scatter(x[:50, 0], x[:50, 1], x[:50, 2], color='red',
    ↪  marker='o', label='setosa')
    # ax.scatter(x[50:100, 0], x[50:100, 1], x[50:100, 2], color='blue',
    ↪  marker='x', label='versicolor')
    # ax.set_xlabel(xlabel_text)
    # ax.set_ylabel(ylabel_text)
    # ax.set_zlabel(zlabel_text)
    # ax.legend()
    # plt.show()

    perceptron = Perceptron(.1, 100)
```

```
    print(x.shape)
    perceptron.fit(x, y)
    #print(perceptron.get_weights())

    plot_decision_regions(x, y, perceptron)
    plt.xlabel(xlabel_text)
    plt.ylabel(ylabel_text)
    plt.show()
# Press the green button in the gutter to run the script.


if __name__ == '__main__':
    main()

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

A.2 Perceptron

```
import numpy as np


class Perceptron:
    def __init__(self, scale, max_epochs):
        self._scale = scale
        self._max_epochs = max_epochs
        self._weights = []

    def _predict(self, x):
        summation = self.net_input(x)
        summation = np.where(summation > 0, 1, -1)
        return summation

    def net_input(self, x):
        return np.dot(x, self._weights[1:]) + self._weights[0]

    def fit(self, x, y):
        self._weights = [0 for index in range(len(x[0])+1)]

        for epoch in range(self._max_epochs):
            error_rate = 0

            for index in range(len(x)):
                point = x[index]
                prediction = self._predict(point)
                error = y[index] - prediction
                error_rate += error ** 2
                self._weights[0] += self._scale * error
```

```
                self._weights[1:] += self._scale * error * point
            print(error_rate)
            if error_rate == 0:
                break

    def get_weights(self):
        return self._weights

    def predict(self, x):
        size = x.shape
        size = size[1]
        retval = self._predict(x)
        return retval
```