Dong (Bob) Lee

02/01/18

Machine Learning Independent Study

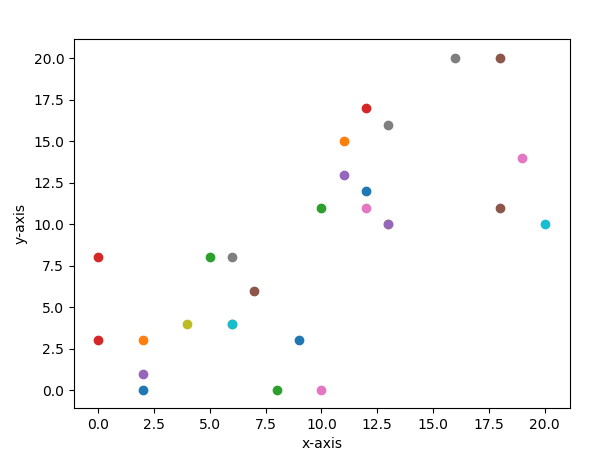
**Abstract/Motivation**

When it comes to any field of study, it is ideal to gradually increment towards to complex concepts, by starting out with fundamental concepts. Therefore, there is a necessity to start out the journey of machine learning by tackling the idea of supervised learning. Before delving into the reasoning why supervised learning is the right way to start the journey of machine learning is, there is a need of comparing the two ideas of machine learning: supervised and unsupervised.

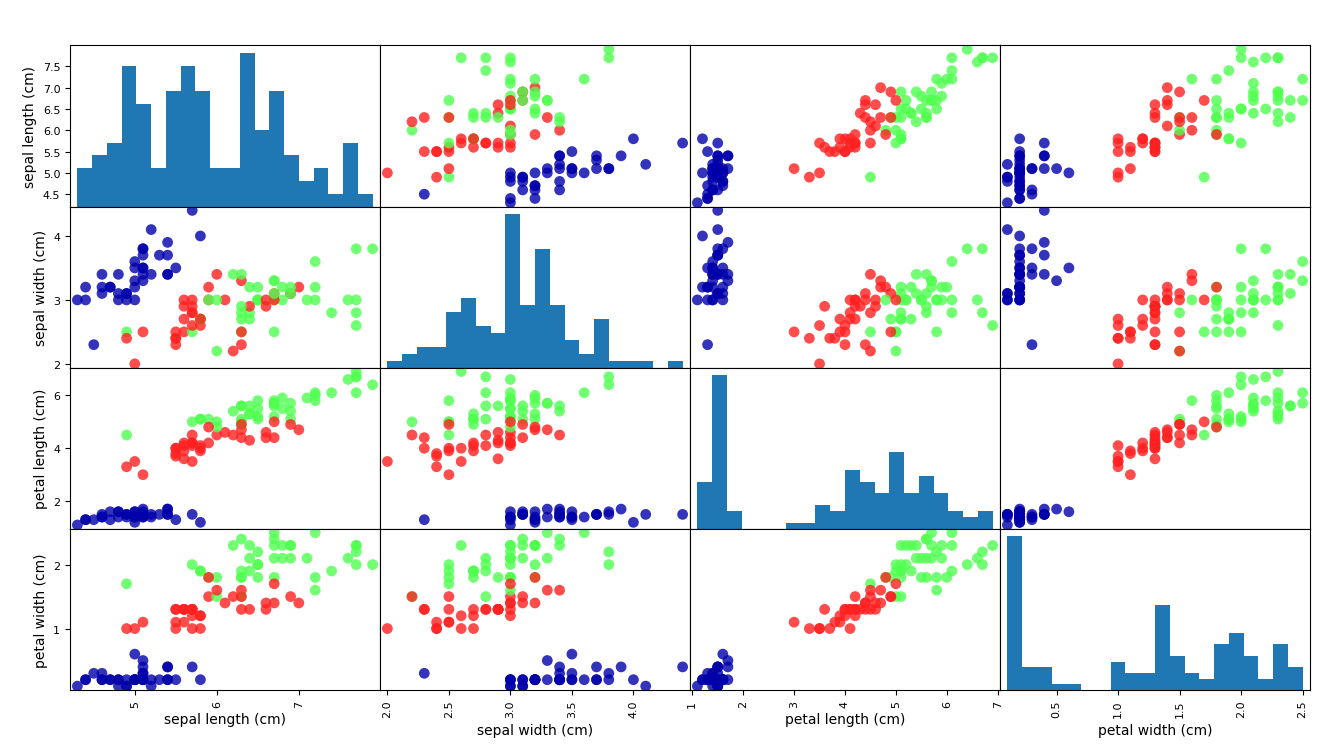
Supervised machine learning is where we already know how the inputs and outputs are expected to look like, and when the new data is introduced to the supervised learning algorithm, machine will be able to predict how the data is needed to be categorized by comparing the pool of data which was given to the algorithm as an input. To summarize what supervised machine learning is, we are pretty much telling the machine that, “since you are more accurate than my human brain is, why won’t you tell me what this data that I am trying to predict by comparing subtle differences this data has with other data I will give you (Müller)?”

Unsupervised learning, however, we don’t know what the right output is; if so, how do we know if the model did well? In the unsupervised learning algorithms, it is similar as telling the machine to, “think instead of me;” therefore, the unsupervised learning algorithm is used to explore the difficult concept, where the difficulty of implementing an application that utilizes the unsupervised learning algorithm spikes compared to the supervised learning algorithm. In this paper, we will go over the ideas of K-Neighbor Classifier and LinearSVC, where those two ideas will be used for our application to decide which classification the data should belong to (Müller).

**Introduction**

Let’s assume we have a pool of data, when plotted, which should look like this: 

There is a clear segregation demonstrated in the plot, but our objective is to have our algorithm recognize that, not us, because when we are faced with a more complex pool of data like below:



Segregation between the data points is still present, but we can notice that the difficulty of deciphering the differences between categories of the data has spiked, especially between green and red data points. At the same time, it would be a great help if we have a regression between the data for clarity purpose; therefore, we will aim to do just that, too. In order to achieve the objective mentioned, we’ll be using matplotlib, numpy, and learn.

**Method/measurement**

For this objective to happen, I’ve referred to Introduction to Machine Learning with Python for layout of how my learning will happen, while referring to Understanding Support Vector Machine, written by [Aurélien Géron](https://www.safaribooksonline.com/search/?query=author%3A%22Aur%C3%A9lien%20G%C3%A9ron%22&sort=relevance&highlight=true), in order to understand what’s happening behind the LinearSVC module, and referred to SciPy and Numpy, written by Eli Bressert, in order to understand the modules I am using on a fundamental level. Sklearn module has a handful of useful datasets we could; however, the purpose of this project is to understand how to utilize the encapsulated modules to achieve the desired result; therefore, we’ll be using the code next page:

import random

def generateInput (iterAmt):

someList = []

for x in range(0, int(iterAmt/2)):

orderedPair = [random.randint(0, 10), random.randint(0, 10)]

someList.append(orderedPair)

for y in range(0, int(iterAmt/2)):

orderedPair = [random.randint(10, 20), random.randint(10, 20)]

someList.append(orderedPair)

return someList

Where the generated input will prepare a list, which will contain integer values from 0 to 10, and integer values ranging from 10 to 20, and return the list to any program that is calling this function.

Also, we must prepare our target values, for this paper, I’ve prepared the target values of:

targetValues = [[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]]

Target values are needed for this project because those integer values will be used to classify certain data points. Now that we have training data, and target data, we are ready to fit our data into the algorithm; for the purpose of consistency in representation for this paper, the implementation will be in the next page.

from sklearn.neighbors import KNeighborsClassifier as Kn

def startProg(predictThis, input, target):

knn = Kn(n\_neighbors=1)

knn.fit(input, target)

result = knn.predict([predictThis])

print("predict() returned: ", result)

From the sklearn.neighbors module, we’ll need KNeighborsClassifier in order for a portion of our program to complete. startProg() function will take in 3 arguments, where predictThis is a random data point, which will be predicted to see how it will be classified, input argument represents the training data, and target argument represents the target data.

KNeightborsClassifier is an object encapsulates k-nearest neighbors algorithm, where k-nearest neighbors algorithm returns a classification of a particular data point which was given to it. In my case, since, I am passing 1 for the n\_neighbors parameter, the algorithm will look for a single closest point to decipher which classification does the data point belong to. The fit() function takes in 2 arguments, in this case, we have input as our training data, and target as out targetValues. Predict() passes a representation of a data point, which will be used in the algorithm to see how it should be classified. The next page will run the implemented program so far at the command prompt, and contain the full display of implemented program.

targetValues = [[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]]

inputValues = inputGenerator.generateInput(len(targetValues))

def startProg(predictThis, input, target):

knn = Kn(n\_neighbors=1)

knn.fit(input, target)

result = knn.predict([predictThis])

print("predict() returned: ", result)

startProg([0, 9], inputValues, targetValues)

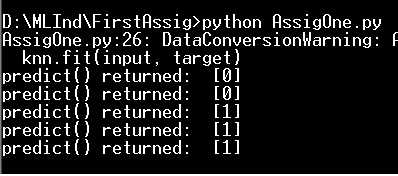
startProg([4, 7], inputValues, targetValues)

startProg([11, 19], inputValues, targetValues)

startProg([12, 17], inputValues, targetValues)

startProg([0, 20], inputValues, targetValues)

The above code will produce this result:



In this case, 0 represents all values ranging from 0 to 10, and 1 represents all values ranging from 10 to 20; just as expected the input values co-align with the results, except the very last input, where [0, 20] not belong to any classification; due to this reason, many will be questioning if the answers are truly correct without any visual representation; in order to resolve this conundrum, we’ll be plotting the data we’ve inputted.

import matplotlib.pyplot as plt

def plotValues(plotThis):

for counter in range(0, len(plotThis)):

plt.scatter(plotThis[counter][0], plotThis[counter][1])

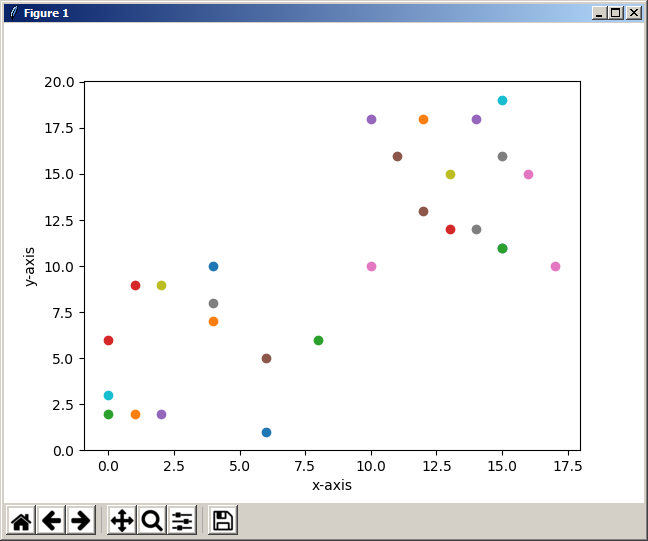
plt.xlabel("x-axis")

plt.ylabel("y-axis")

plt.show()

plotValues(inputValues)

The code above will pass in a two-dimensional array, which will be our data points. The scatter() function will plot the data points in without plotting a continuous line, where each plotted points are separate from each other, and put.show() will display the graph, where we should be able to see image being produced, demonstrated in the next page.



In one of the function we have invoked:

startProg([0, 20], inputValues, targetValues)

we can notice that [0, 20] shouldn’t be classified with any integer; we could throw an exception, but there is a much more visual way to catch irregularities like above, we could display a regression separating two different categories of points.

from sklearn.svm import LinearSVC

def plotRegression(inputValues, targetValues):

svm = LinearSVC(C=1)

svm.fit(inputValues, targetValues)

w = svm.coef\_[0]

a = -w[0]/w[1]

xx = np.linspace(0, 20)

yy = a \* xx - (svm.intercept\_[0]) / w[1]

for counter in range(0, len(inputValues)):

plt.scatter(inputValues[counter][0], inputValues[counter][1])

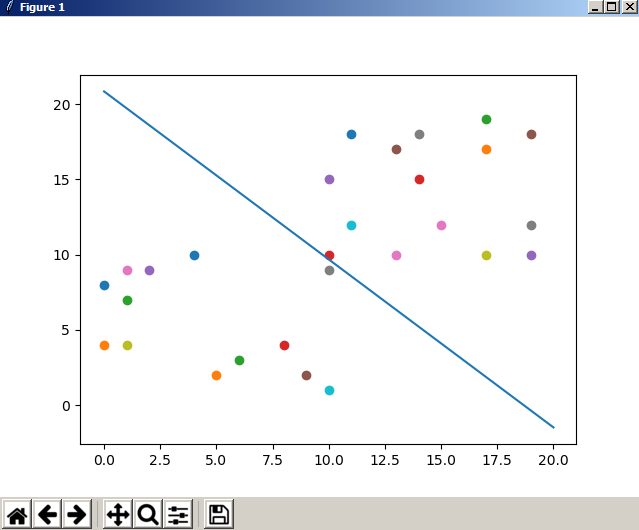
plt.plot(xx, yy)

plt.show()

LinearSVC object encapsulates support vector machine when manipulated as stated in the below documentation,

<http://scikit-learn.org/0.18/auto_examples/svm/plot_separating_hyperplane.html>

we’ll be able to see a nice regression between the points when we plot it. The linspace() returns an evenly spaced value in an array, and we’ll pass in xx and yy in the plot(), which will display the linear line in the Cartesian graph. When we invoke this function, we should be able to see graph below:



The regression above allows us to see the separation between the data points, where representation such as this utilizes the accuracy of the machine; while, at the same time, represents the data which can be easily conceived by human perception.

**Results**

After these activities, one could get the gist of what supervised learning could look like; and, with further studies in the matter, one could have much more solidified intuition that supervised learning could solve problems where human perception becomes near useless.

**Conclusion**

The conclusion is that in Supervised Learning algorithms, the outputs needs to be still sensible to the human perception; meaning, the Supervised Machine Learning Algorithms are for the purpose of analyzing data, not necessarily discovering new data.

**Feedback for instructor**

Understanding the mathematical portion of this assignment is still not done, it is true that encapsulation exists to avoid just that, but, non-the less it was the most difficult portion of this project.

I have a hard time foreseeing the possibility assignment becoming any more easier, scipy does most of the work for me; however, it does not mean the assignment was not difficult, I’d like to express that everything was encapsulated very well.

I learned how to maneuver around math-centric modules in Python, and became more comfortable with their documentation than when I initially started this assignment.

I’ve referred to many of the learning sources in Safari, provided by the O’reily publisher.

**Bibliography**

**Formal**

Müller, Andreas C., and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. OReilly, 2017.

Géron, Aurélien. *Understanding support vector machines*. O'Reilly Media, Inc., 2017.  
Bressert, Eli. *SciPy and NumPy*. OReilly, 2013.

**Informal**

“SVM: Maximum margin separating hyperplane¶.” *SVM: Maximum margin separating hyperplane - scikit-Learn 0.18.2 documentation*, scikit- learn.org/0.18/auto\_examples/svm/plot\_separating\_hyperplane.html.