Dong (Bob) Lee

02/13/18

Independent Machine Learning Assignment 2

**Abstract/Motivation**

The paper goes over the idea of unsupervised machine learning algorithms (where this paper will go over the principal component analysis), and techniques of scaling values, where the scaling techniques compliments unsupervised machine learning algorithms by increasing the accuracy. The paper will also go over the compare and contrast between the supervised machine learning algorithm, and the unsupervised machine learning algorithm.

**Introduction**

In many instances of data gathering phase, the data being gathered often turns out to be unusable in order to solve problems, and more often than not, there need to be rescaling of the data, in order to maintain the accuracy of the machine learning algorithms; therefore, this paper will go over the preprocessing libraries. One could intuitively argue that, "there is no need to rescale the data, if the scientist already knows how the output should look like," such way of thinking is certainly the case, but unsupervised learning is a machine learning algorithm certainly provides an extracted data according to the given input, but they do not have any labeling; thus, in much-unsupervised learning algorithms, the scientist must exam the output, and decipher if it is even what they were looking for. Chaining to the previous concept that was mentioned, unsupervised learning could take a greater advantage of scaling, better than the supervised learning, and the reason is because in supervised learning, the programmer is implementing the algorithms, already knowing how each prediction will look like; therefore, accuracy of the data in unsupervised learning must be in bigger concern than the supervised learning. Even so, since, unsupervised learning machine learning algorithm gives the users the output, where we do not know if the answer is correct; therefore, it should be only used in the expletory setting.

**Method/measurement**

The preprocessing in the machine learning is categorized under an unsupervised transformation of a dataset, where a particular dataset becomes mutated, in order to be suitable for an algorithm, or things alike. For example, Scikit API StandardScaler takes in a set of data, which will ensure that for each feature the mean is 0 and the variance 1 (Introduction to Machine Learning with Python). Another example could be MinMaxScaler, where MinMaxScaler takes in a dataset and mutates the all the values in between the 0’s and 1’s. The above procedures can be categorized as a rescaling of the data. Let’s assume we have a data set for of cancer, we could import it from the sklearn.dataset, where the code would look like this:

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_breast\_cancer as cancer

X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target,

random\_state=1)

In the next page, there will be a demonstration of SVC’s accuracy, with the current data set provided in the above code.

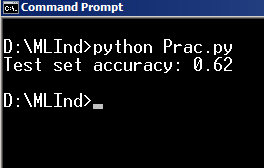
from sklearn.svm import SVC

svm=SVC(C=100)

svm.fit(X\_train, y\_train)

print("Test set accuracy: {:.2f}".format(svm.score(X\_test, y\_test)))

From the above sources,



The machine learning model we currently have maintains the accuracy of 0.62, and with the power of preprocessing, there can be a drastic change in accuracy with the below code:

from sklearn.svm import SVC

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

cancer = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target, random\_state=1)

svm = SVC(C=100)

scaler = MinMaxScaler()

scaler.fit(X\_train)

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

svm.fit(X\_train\_scaled, y\_train)

print("Scaled test set accuracy: {:.2f}".format(

svm.score(X\_test\_scaled, y\_test)))

When this program runs, the console outputs:



Compare to the previous accuracy, that’s a whopping 31 percent increase in accuracy.

With the know-how to possibly increase the accuracy of machine learning model, it is safe to conclude that it’s time to move on to the unsupervised learning algorithm. Like the introduction of this paper has mentioned, the paper will utilize the Principal Component Analysis. PCA (principal component analysis) is a compression tool, which enables the user to compress multi-dimensional data into a smaller dimension. For example, referring back to the breast\_cancer dataset, with the code demonstrated below:

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

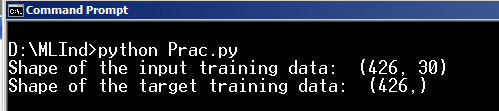
cancer = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target,

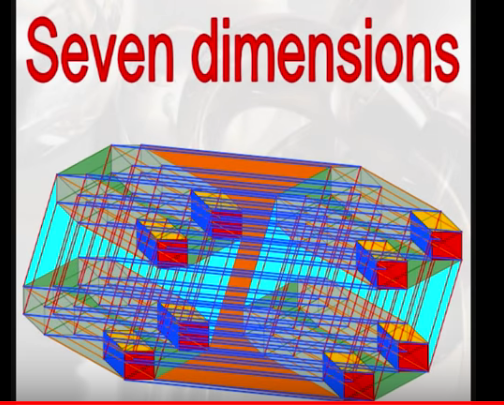
random\_state=1)

print("Shape of the input training data: ", X\_train.shape)

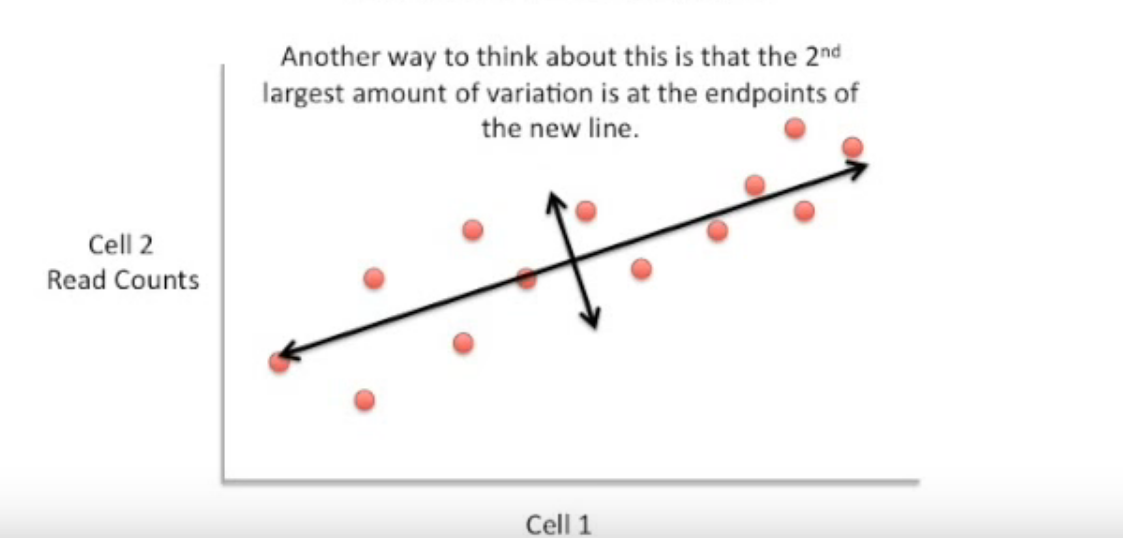
print("Shape of the target training data: ", y\_train.shape)



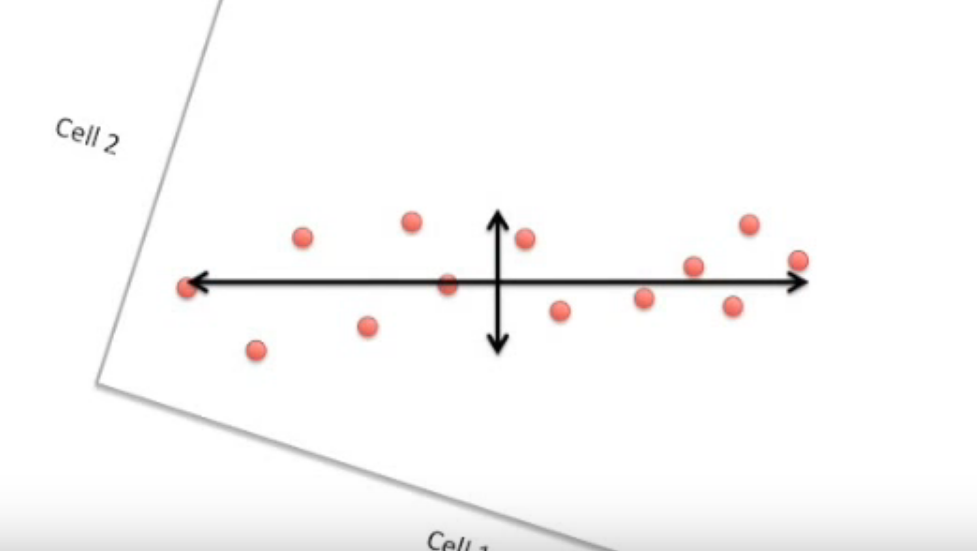
We’ll be able to see this resultant, where breast cancer input training data is logically equivalent to 426 by 30 matrix. In a situation like this, in order for this data to be even visually digestible (because visually representing anything beyond 6th dimension becomes almost impossible to graph), there needs to be a compression of some sort, unless one wants to work their way around to represent data points in a graphical representation, comparable to the picture in the next page:



Therefore, an algorithm like PCA is needed to compress the data to be visually "human digestible." PCA works because of this concept called principle components; principal components are groups of orthogonal vectors, sprouting from the most concentrated area of the data when looked at it from the graphical perspective. To put it generally, PCA takes dataset with a lot of dimensions, and be able to flatten it to 2 or 3 dimensions. PCA rotates the dataset, such that the Cartesian graph’s origin appears as if it’s in the center of the most concentrated portion of the data; meaning if the data points are spread out like the picture in next page:



PCA would make it so that the data points will be represented in angled perspective, like:



where the principle components behaves as an x-axis and y-axis, in the case above (joshstarmer).

The block of code below demonstrates of PCA objects can be utilized:

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

scaler = StandardScaler()

scaler.fit(cancer.data)

X\_scaled = scaler.transform(cancer.data)

pca = PCA(n\_components=2)

pca.fit(X\_scaled)

X\_pca = pca.transform(X\_scaled)

print("X\_pca is: ", X\_pca)

plt.figure(figsize=(8, 8))

The block of code above scales the dataset of breast cancer to increase the accuracy, where the X\_scaled is assigned with the scaled set of data. The variable pca is assigned with the PCA object, where the n\_components represents the principal components. The above program outputs:



Upon transpose, the matrix can be represented in a 2-dimensional graph. The code in the next page is a program that displays the plot of compressed version of the breast cancer dataset in to a plot, by utilizing the functions provided by Matplotlib, Sklearn, and mglearn (which is a package provided by author of the Introduction to machine learning with Python: a guide for data scientists). The Matplotlib API will be used for plotting data points that are compressed from the PCA object, Sklearn will be used to implement the actual machine learning algorithms, and mglearn will be used to complement the graph that will be displayed.

import matplotlib.pyplot as plt

import mglearn

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target, random\_state=1)

scaler = StandardScaler()

scaler.fit(cancer.data)

X\_scaled = scaler.transform(cancer.data)

pca = PCA(n\_components=2)

pca.fit(X\_scaled)

X\_pca = pca.transform(X\_scaled)

plt.figure(figsize=(8, 8))

plt.legend(cancer.target\_names, loc="best")

mglearn.discrete\_scatter(X\_pca[:, 0], X\_pca[:, 1], cancer.target)

plt.legend(cancer.target\_names, loc="best")

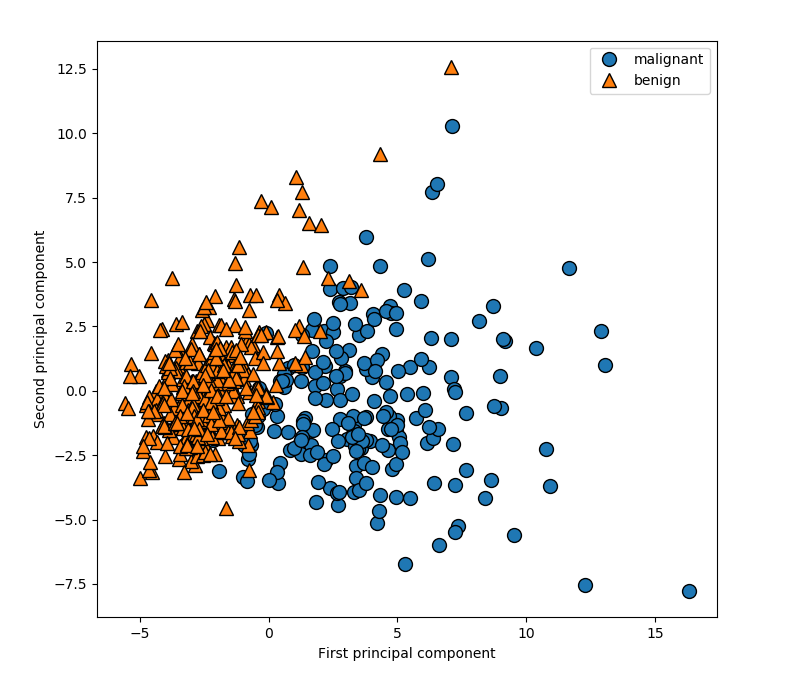
plt.gca().set\_aspect("equal")

plt.xlabel("First principal component")

plt.ylabel("Second principal component")

plt.show()

Aside from the last block of the code above everything else was introduced before, where the last block of the code above finalizes things by making the plot look pretty; however, mglearn is a library provided by the author, where the discrete\_scatter plots the data point with different colors, (which is the third argument of the function in this case). The actual work is being done in the PCA object, where X\_scaled data is passed into the pca.fit(). When the above program runs, we should be able to see something along the line of like the picture on the next page (Müller).



Originally, the dataset regarding the breast cancer was 426 by 30 matrix (after the test train split), but with the power of PCA, the graphical representation can be condensed to whopping 2 by n matrix, where n is the number of columns, after the transpose for the array.

**Result**

Implementation of PCA proves to be powerful; however, one needs to recognize that this was entirely done without any supervision, the data that was returned from the pca.fit(), in the previous blocks of code, was entirely created by PCA object; hence, this type of learning is called unsupervised learning. When we were dealing with the dataset of breast cancer, looking at the initial data, there was near to no way of deciphering which data points should be considered malignant, or benign. Looking at the plot, one could still be thrown into a confusion, for certain data points collapse together, despite being categorically different from each other (where, in a case like this, regression is impossible, and so is Kth-Algorithm), where we let the PCA decide if the data point is truly benign, or malignant.

**Bibliography**

**Formal**

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

**Informal**

joshstarmer. “StatQuest: Principal Component Analysis (PCA) clearly explained.” *YouTube*, YouTube, 13 Aug. 2015, [www.youtube.com/watch?v=\_UVHneBUBW0](http://www.youtube.com/watch?v=_UVHneBUBW0).

“What is Data Preprocessing? - Definition from Techopedia.” *Techopedia.com*, www.techopedia.com/definition/14650/data-preprocessing.

**Feedback for instructor**

**What was too difficult, too easy?**

Nearly every part of this “run” was ridiculously easy because libraries nicely encapsulate everything about scaling and PCA, and concepts, in an abstract sense, isn’t that hard to understand either.

**What would have made the learning experience better?**

Personally, I wish the programs I write in the machine learning independent study is more geared towards to writing things in system programming; from now one, I am going to try my best to do just that.

**What did you learn?**

The concrete idea that machine learning is still largely related to data science, where it’s a study of the field, which is further away from the purity of computer science. The idea of PCA solidifies that, for PCA in itself has an origin from a field of a statistic.

**How did you learn it?**

Implementing programs regarding unsupervised learning, and comparing my previous experiences with the learning experiences I gain from programming machine learning things.