Machine Learning Engineer Nanodegree

Capstone Project

07.13.2018

I. Definition

Project Overview

Recently, many works have been performed depending on the computer. For the processing time to be reduce and provide more accurate results, for example, depending on different types of data, such as characters and digits are used often during our life cycle. To automate the systems that deal with such figures as the postal code, numbers of bank accounts and numbers on car plates.

The technology of character recognition was due to the increase requirement to convert a huge amount of printed or handwritten information to digital format. This conversion from paper to computer in the past required a person operators who processed billions of checks, mail correspondence, etc. This process was laborious and error-prone, motivating the development of a method, which is for reading data and recognizing one character after other.

In general machine learning algorithms, inspired by psychology and biology, which studies the data set and can be used to solve digit recognition and wider problems. A controlled model of machine learning is provided to instances of data specific to a problem domain and a response that solves the problem for each instance. When training a full model can not only provide answers to data that it has studied, but also to unprecedented data with high accuracy.

From digit recognition aspect, studies can be divided into two main groups with regard to input characteristics: The first group consists of studies using numbers obtained as inputs and the second - studies using the entire image as input data.

One of the most important parts of object recognition algorithms and recognition of handwritten figures algorithms are a classification. Classification in computer Science is a prediction of a class or a label for an object based on its similarity to previous objects. In the process of machine learning, each object or instance represented by the same set of functions. Based on algorithm of training, classifiers can be divided into unsupervised and supervised classifiers. Supervised learning uses the knowledge of labels for instances is used to build a model, while instances for unsupervised learning is not labeled.

In this project, I worked on a python application of recognizing handwritten digits from UCI Repository lab – Semeion dataset. The problem taken into account as a classification problem and SVM algorithm is used.

Problem Statement

The goal of this project is to create a model that can recognize and define the handwritten digits using the Support Vector Machine.

Support Vector Machine was offered by Vapnik as a binary categorizer (https://en.wikipedia.org/wiki/Support_vector_machine). SVM detects a hyperplane that separates data from different classes. Each instance is marked with one of the existing classes and they are represented as points in space. SVM builds a model based on instances from training set and further classification of unknown instances is done by that model.

As data set Semeion Handwritten Digit Data Set is used which has 1593 handwritten digits from around 80 persons were scanned, stretched in a rectangular box 16x16 in a gray scale of 256 values. Then each pixel of each image was scaled into a bolean (1/0) value using a fixed threshold.

Each person wrote on a paper all the digits from 0 to 9, twice. The commitment was to write the digit the first time in the normal way (trying to write each digit accurately) and the second time in a fast way (with no accuracy).

Metrics

It will be given as a simple measurement of accuracy as a general measure of models performance. This will give an estimate that takes into account all the right and wrong classifications. It will be as given below,

accuracy = # correctly predicted digits / # total number of digits

A high score will reflect a large number of correctly predictions and correctly predicted as a percentage of the total number of attempts at predictions. As I worked as a classification problem, this metric provided reasonable justification for the problem.

II. Analysis

Data Exploration

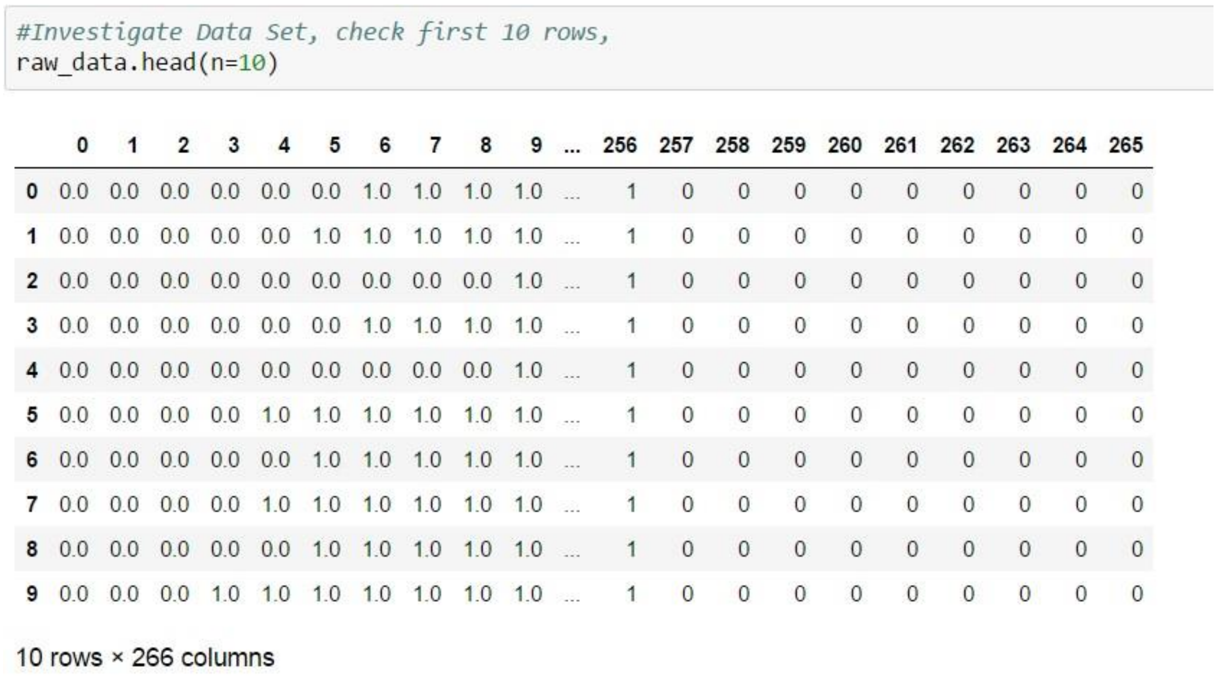
Semeion Handwritten Digit Data set consists of 1593 records (rows) and 256 attributes (columns).

Each entry is a handwritten digit originally scanned with a 256 gradient scale resolution (28).

Each pixel of each original scanned image was first stretched, and after scaling between 0 and 1 (setting to 0 for each pixel whose value was under the 127 value of the gray scale (127 enabled) and setting to 1 each pixel whose value in the gray scale was more 127).

257-265th columns are given as labels.

For the first 10 entry is given below,



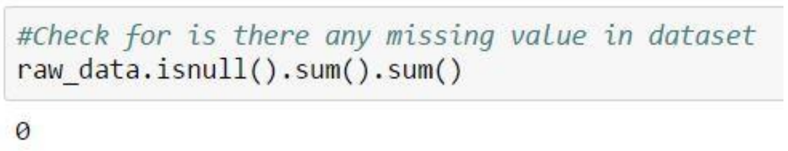
```python
#Investigate Data Set, check first 10 rows,
raw_data.head(n=10)
```

	0	1	2	3	4	5	6	7	8	9	...	256	257	258	259	260	261	262	263	264	265
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
1	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	1	0	0	0	0	0	0	0	0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	1	0	0	0	0	0	0	0	0	0
5	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
6	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
7	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
8	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0
9	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1	0	0	0	0	0	0	0	0	0

10 rows × 266 columns

Figure-1: Dataset overview

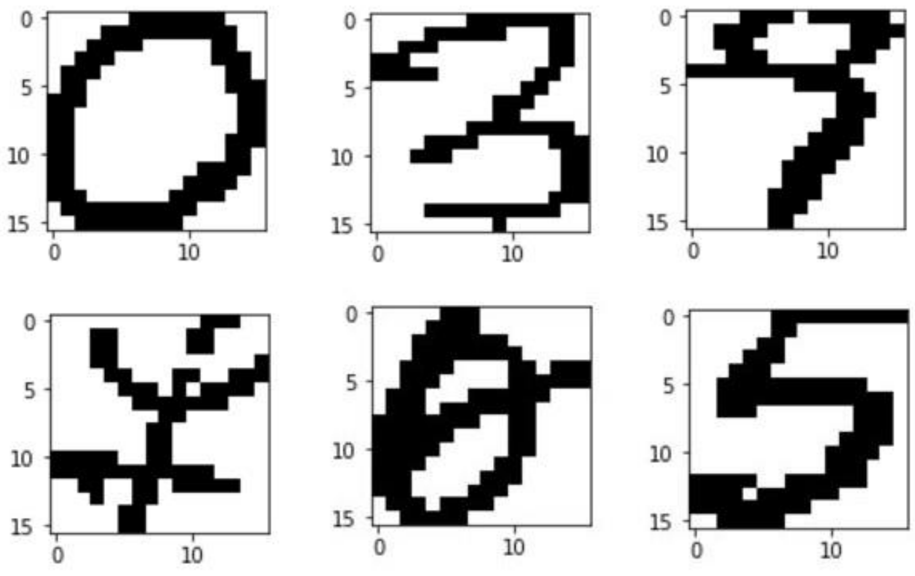
Dataset has so many columns, so it is needed to be checked for missing data and observed that there is no missing data in dataset.



```python
#Check for is there any missing value in dataset
raw_data.isnull().sum().sum()
```

```
0
```

Figure-2 show some example digits from dataset,



Exploratory Visualization

We could check distribution of labels below,

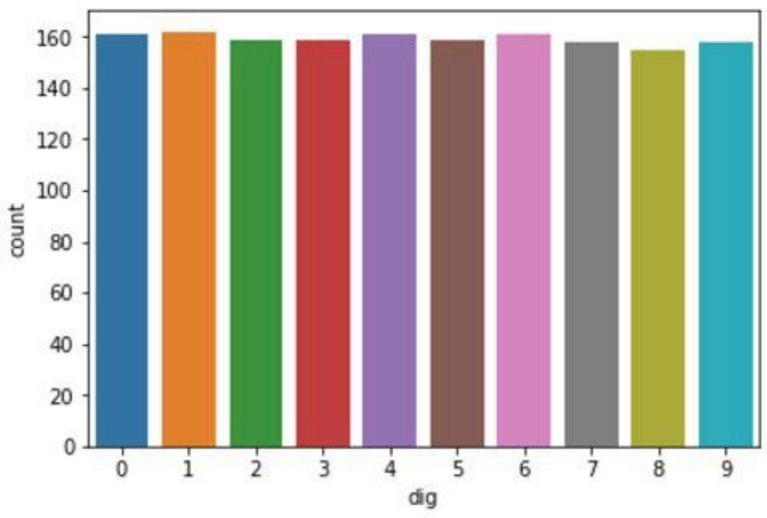


Figure 3- Label Distribution

X axis shows us digits (0 to 9) and Y axis shows us total number of digits which are exist in dataset. It is clearly seen that digits have equal counts in dataset. This is important for our analyses. If dataset do not have equal number of digits, it can negatively affect our algorithm.

Algorithms and Techniques

In this project, I used SVM with PCA for digit recognition.

SVM is a powerful machine learning tool capable of representing nonlinear relationships and creating models that are well summarized for invisible data. For binary classification, linear SVM (the simplest form of SVM) finds the optimal linear separator between two data classes. This optimal separator is the result of in the widest limit of the separation between the two classes, since the wide passage implies that the classifier is better able to classify unseen data. To regulate retraining, SVM have a parameter of complexity, C, which determines the trade-off between choice a large classifier and the amount for which erroneously classified samples are allowed. A higher value of C means that a greater value is given to minimizing the sum erroneous classification than the search for a model with a large margin. For the processing of nonlinear data (for example, radial base function (RBF), polynomial or sigmoid) are introduced in match the original data with a new object space in which you can find a linear separator. In addition to the parameter C, each kernel can have a number of related parameters with this. In the project here , linear, rbf, sigmoid and poly kernels are used which has no additional parameter (only poly degree changed to 2 from default value 3). In general, SVM is considered useful for processing high dimensional data.

PCA is a classic statistical method for converting the attributes of a data set into a new one a set of uncorrelated attributes called main components. PCA can be used for reduce the dimension of the data set, while retaining most of the variability a set of data, as far as possible. High-dimensional data can create problems for machine learning since predictive models based on such data are at risk of retraining. In addition, many attributes can be redundant or highly correlated.

The accuracy of the SVM classification is very sensitive to the quality of the training set. To avoid the direct use of complex and high-dimensional coefficients, I bring the analysis of the main components PCA into the SVM model to extract low-dimensional and effective information about objects, which increases the accuracy and effectiveness of training, and also preserves the original data features.

Benchmark

The benchmark model here is again SVM without applying PCA with the same dataset with same features.

III. Methodology

1. Data preparation (Described in Methodology III- Data processing section)

- Load data
- Check for abnormalities
- Separate the dataset into images and labels
- Visualization

2. SVM-PCA (Described in Methodology III- Implementation & Refinement section)

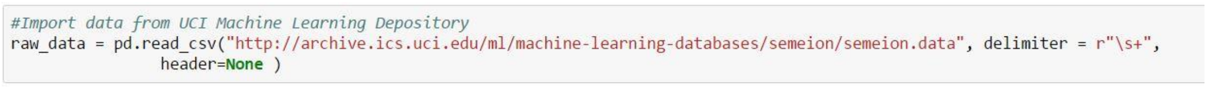
- Splitting data for training and testing
- Run SVM with four kernels (linear,rbf,sigmoid,poly)
- Dimensionality Reduction using PCA
- Define the model with SVM PCA linear kernel

3. Evaluate the model (Described in Results IV- Model Evaluation and Validation section)

- Get accuracy scores
- Visually check false predictions

Data Preprocessing

Data loaded directly from UCI Repo,

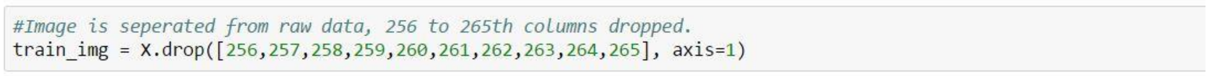


```
#Import data from UCI Machine Learning Depository
raw_data = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/semeion/semeion.data", delimiter = r"\s+",
                 header=None )
```

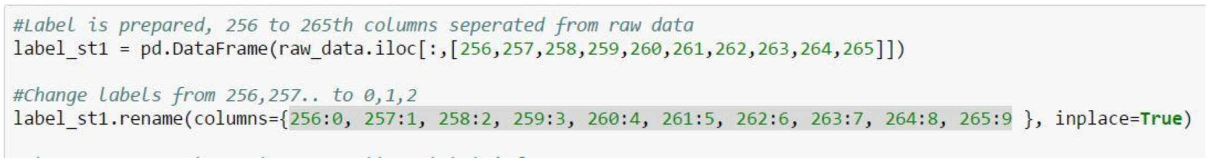
In raw data (Whole data set) has 256 (16 X 16) columns (0 to 255) of floats containing 0 and 1 that will be used as input variables (train_x). Images of handwritten figures are converted to monochrome, so there is only two colors & black and white. This simplifies our approach to determining numbers to colors and gradients. It is easily assigned 1 to black, and from 0 to white.

In raw data (whole data set) 10 columns (256 to 265), from 0 to 9 representing the label of variable (train_lab).

Firstly, inputs and labes are separated. Labes are assigned 256:0, 257:1, 258:2, 259:3, 260:4, 261:5, 262:6, 263:7, 264:8, 265:9 as given below,

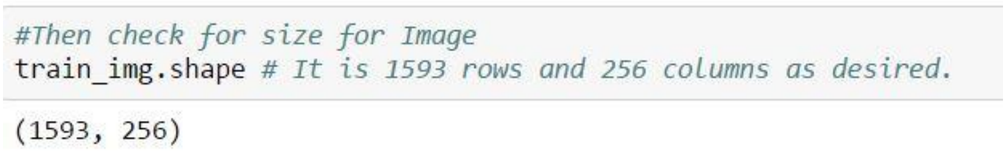


```
#Image is seperated from raw data, 256 to 265th columns dropped.
train_img = X.drop([256,257,258,259,260,261,262,263,264,265], axis=1)
```

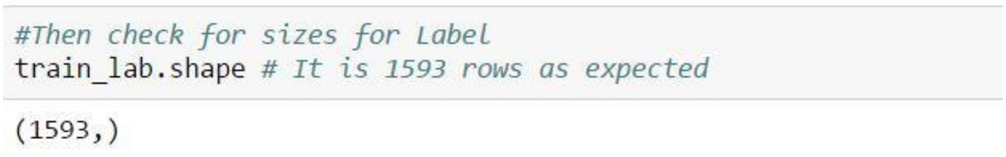


```
#Label is prepared, 256 to 265th columns seperated from raw data
label_st1 = pd.DataFrame(raw_data.iloc[:,[256,257,258,259,260,261,262,263,264,265]])

#Change labels from 256,257.. to 0,1,2
label_st1.rename(columns={256:0, 257:1, 258:2, 259:3, 260:4, 261:5, 262:6, 263:7, 264:8, 265:9 }, inplace=True)
```

Therefore, one dataset is divided two sets as inputs and labels. Then dimensions are checked,



```
#Then check for size for Image
train_img.shape # It is 1593 rows and 256 columns as desired.

(1593, 256)
```



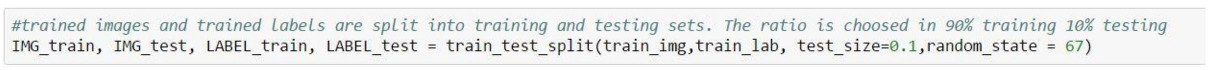
```
#Then check for sizes for Label
train_lab.shape # It is 1593 rows as expected

(1593,)
```

Once seen there is no abnormalities proceeded to implementation.

Implementation

The objective here is to predict given image, and which digit it represents. It is given samples of each of the 10 possible classes (numbers from zero to nine) by which we approach the evaluation in order to be able to predict the classes to which invisible patterns belong.

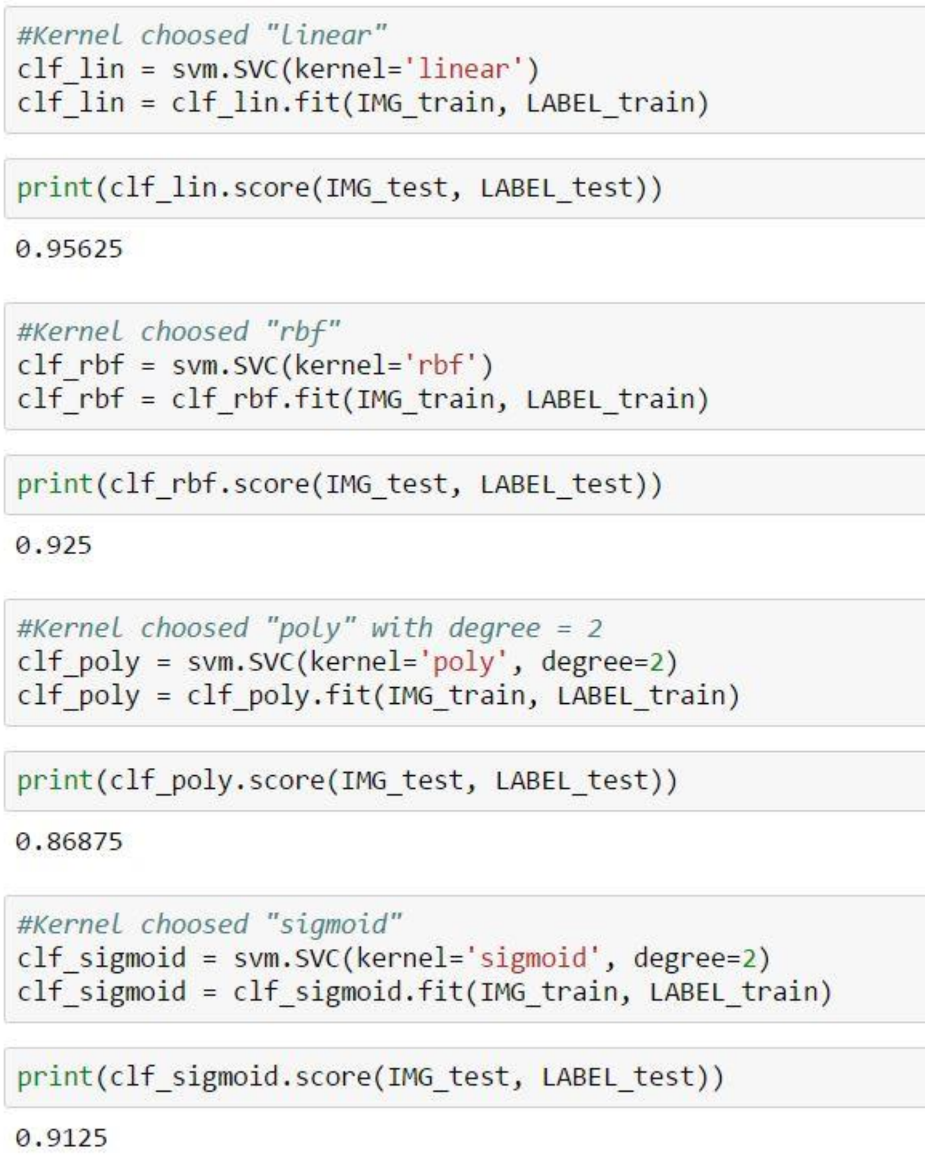
Firstly, image and label data is splitted according to skilearn train_test_split, test size is set 10% and train size is set 90%



```
#trained images and trained labels are split into training and testing sets. The ratio is choosed in 90% training 10% testing
IMG_train, IMG_test, LABEL_train, LABEL_test = train_test_split(train_img,train_lab, test_size=0.1,random_state = 67)
```

It is called that clf evaluation instance, since it is a classifier. Now it must be adapted to the model and learn from the model.

An example of an evaluation is the class sklearn.svm.SVC, which implements vector support classification. In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y). Scores are calculated as given below,



```
#Kernel choosed "linear"
clf_lin = svm.SVC(kernel='linear')
clf_lin = clf_lin.fit(IMG_train, LABEL_train)
```

```
print(clf_lin.score(IMG_test, LABEL_test))
```

```
0.95625
```

```
#Kernel choosed "rbf"
clf_rbf = svm.SVC(kernel='rbf')
clf_rbf = clf_rbf.fit(IMG_train, LABEL_train)
```

```
print(clf_rbf.score(IMG_test, LABEL_test))
```

```
0.925
```

```
#Kernel choosed "poly" with degree = 2
clf_poly = svm.SVC(kernel='poly', degree=2)
clf_poly = clf_poly.fit(IMG_train, LABEL_train)
```

```
print(clf_poly.score(IMG_test, LABEL_test))
```

```
0.86875
```

```
#Kernel choosed "sigmoid"
clf_sigmoid = svm.SVC(kernel='sigmoid', degree=2)
clf_sigmoid = clf_sigmoid.fit(IMG_train, LABEL_train)
```

```
print(clf_sigmoid.score(IMG_test, LABEL_test))
```

```
0.9125
```

Linear SVM is used when have a linearly seperable set of data points. As in 2D, there is set of two-dimensional points that are separated by a straight line. In this case, the linear SVM tries to find a straight line that maximizes the margin to the nearest point to it.

Actually, polynomial kernel is similar, but the boundary is of some defined but arbitrary order

(e.g. order 3: a=b1+b2·X+b3·X2+b4·X3a=b1+b2·X+b3·X2+b4·X3).

The RBF uses normal curves around the data points and summarizes them so that the decision boundary can be determined by the type of topology condition, such as curves where the sum is greater than 0.5.

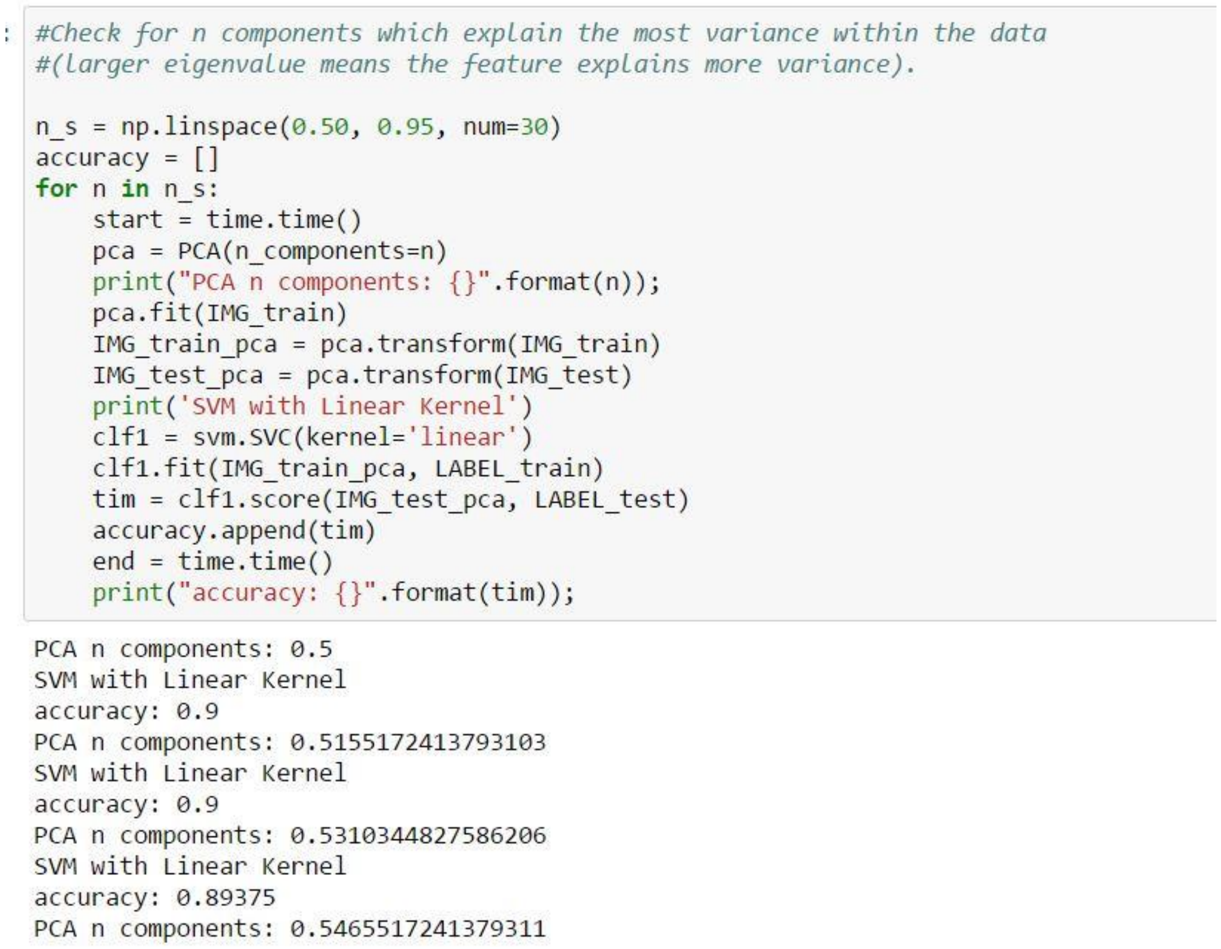
It is interesting to note that the SVM model using the sigmoid function is equivalent to a two-layer perceptron neural network. This core was quite popular for vector support machines because of its origin from the neural network theory.

Since our problem is 2D and linearly separable, to get best results with linear kernel is not surprising.

It is possible to get best results while changing original metrics but I would like to investigate accuracy results after implementing PCA with same metrics. When I changed to gamma coefficient got 1% better results for rbf, poly, sigmoid. There is no change on liner kernel result where these are the parameters for a nonlinear support vector machine (SVM) with a Gaussian radial basis function kernel.

Refinement

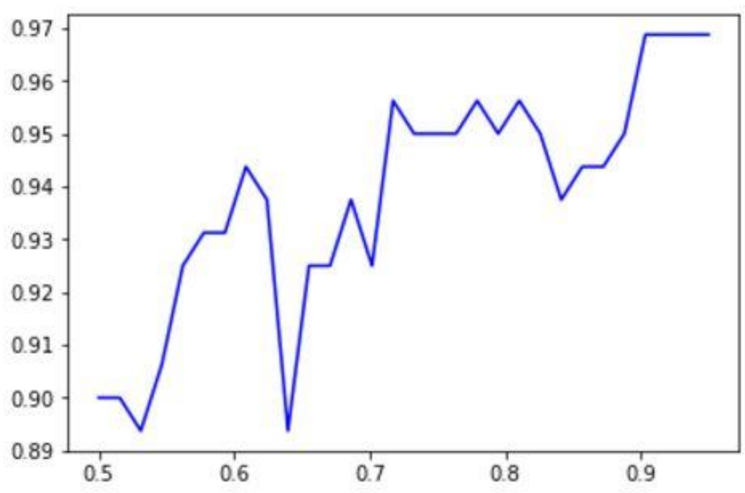
At this step PCA is applied. The curicial point here is n_componenent which explain the most variance within the data. After try some values trying by hand, I decided to check n_component value with a for loop as given below,



```python
#Check for n components which explain the most variance within the data
#(larger eigenvalue means the feature explains more variance).

n_s = np.linspace(0.50, 0.95, num=30)
accuracy = []
for n in n_s:
    start = time.time()
    pca = PCA(n_components=n)
    print("PCA n components: {}".format(n));
    pca.fit(IMG_train)
    IMG_train_pca = pca.transform(IMG_train)
    IMG_test_pca = pca.transform(IMG_test)
    print('SVM with Linear Kernel')
    clf1 = svm.SVC(kernel='linear')
    clf1.fit(IMG_train_pca, LABEL_train)
    tim = clf1.score(IMG_test_pca, LABEL_test)
    accuracy.append(tim)
    end = time.time()
    print("accuracy: {}".format(tim));
```

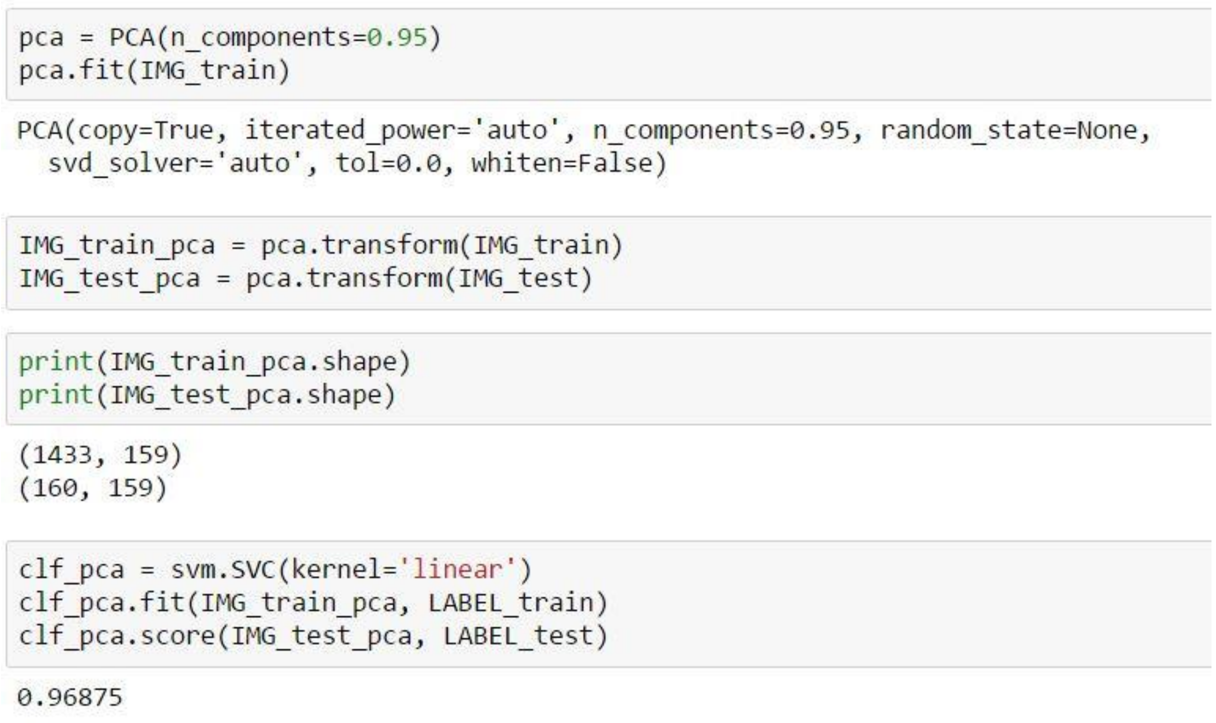
```
PCA n components: 0.5
SVM with Linear Kernel
accuracy: 0.9
PCA n components: 0.5155172413793103
SVM with Linear Kernel
accuracy: 0.9
PCA n components: 0.5310344827586206
SVM with Linear Kernel
accuracy: 0.89375
PCA n components: 0.5465517241379311
```

After that visualize this values in order to choose best value as given below,



For each kernel, same loop is run and according the best n_components value results are taken as given below,

Linear Kernel,



```
pca = PCA(n_components=0.95)
pca.fit(IMG_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
IMG_train_pca = pca.transform(IMG_train)
IMG_test_pca = pca.transform(IMG_test)
```

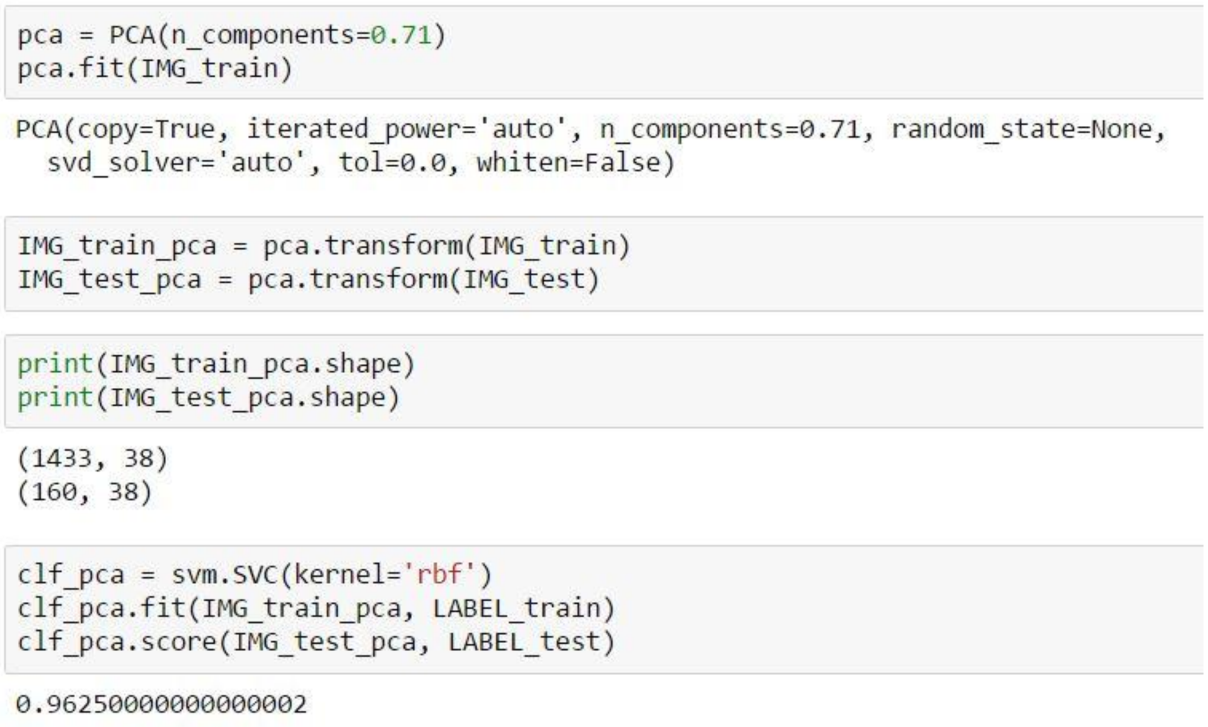
```
print(IMG_train_pca.shape)
print(IMG_test_pca.shape)
```

```
(1433, 159)
(160, 159)
```

```
clf_pca = svm.SVC(kernel='linear')
clf_pca.fit(IMG_train_pca, LABEL_train)
clf_pca.score(IMG_test_pca, LABEL_test)
```

```
0.96875
```

rbf Kernel,



```
pca = PCA(n_components=0.71)
pca.fit(IMG_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.71, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
IMG_train_pca = pca.transform(IMG_train)
IMG_test_pca = pca.transform(IMG_test)
```

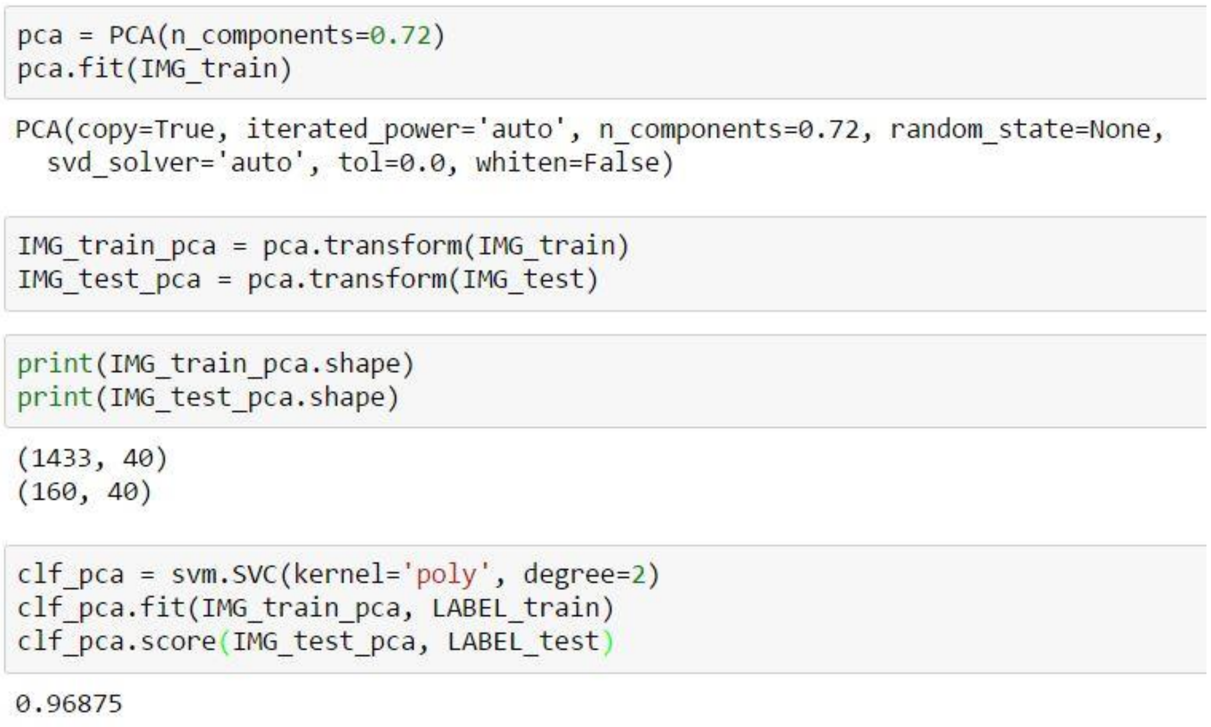
```
print(IMG_train_pca.shape)
print(IMG_test_pca.shape)
```

```
(1433, 38)
(160, 38)
```

```
clf_pca = svm.SVC(kernel='rbf')
clf_pca.fit(IMG_train_pca, LABEL_train)
clf_pca.score(IMG_test_pca, LABEL_test)
```

```
0.9625000000000002
```

Poly kernel (degree=2),



```
pca = PCA(n_components=0.72)
pca.fit(IMG_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.72, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
IMG_train_pca = pca.transform(IMG_train)
IMG_test_pca = pca.transform(IMG_test)
```

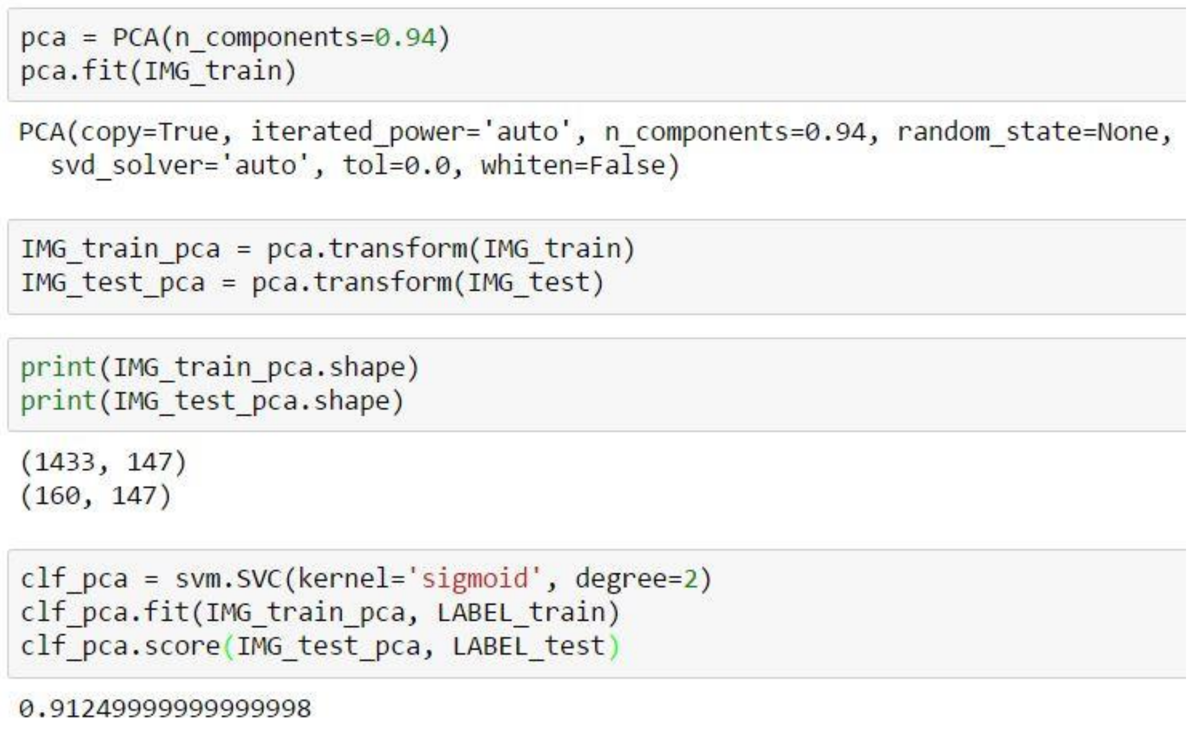
```
print(IMG_train_pca.shape)
print(IMG_test_pca.shape)
```

```
(1433, 40)
(160, 40)
```

```
clf_pca = svm.SVC(kernel='poly', degree=2)
clf_pca.fit(IMG_train_pca, LABEL_train)
clf_pca.score(IMG_test_pca, LABEL_test)
```

```
0.96875
```

Sigmoid kernel (degree=2),



```
pca = PCA(n_components=0.94)
pca.fit(IMG_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.94, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
IMG_train_pca = pca.transform(IMG_train)
IMG_test_pca = pca.transform(IMG_test)
```

```
print(IMG_train_pca.shape)
print(IMG_test_pca.shape)
```

```
(1433, 147)
(160, 147)
```

```
clf_pca = svm.SVC(kernel='sigmoid', degree=2)
clf_pca.fit(IMG_train_pca, LABEL_train)
clf_pca.score(IMG_test_pca, LABEL_test)
```

```
0.91249999999999998
```

As it is expected PCA improves the score however it does not affect sigmoid.

IV. Results

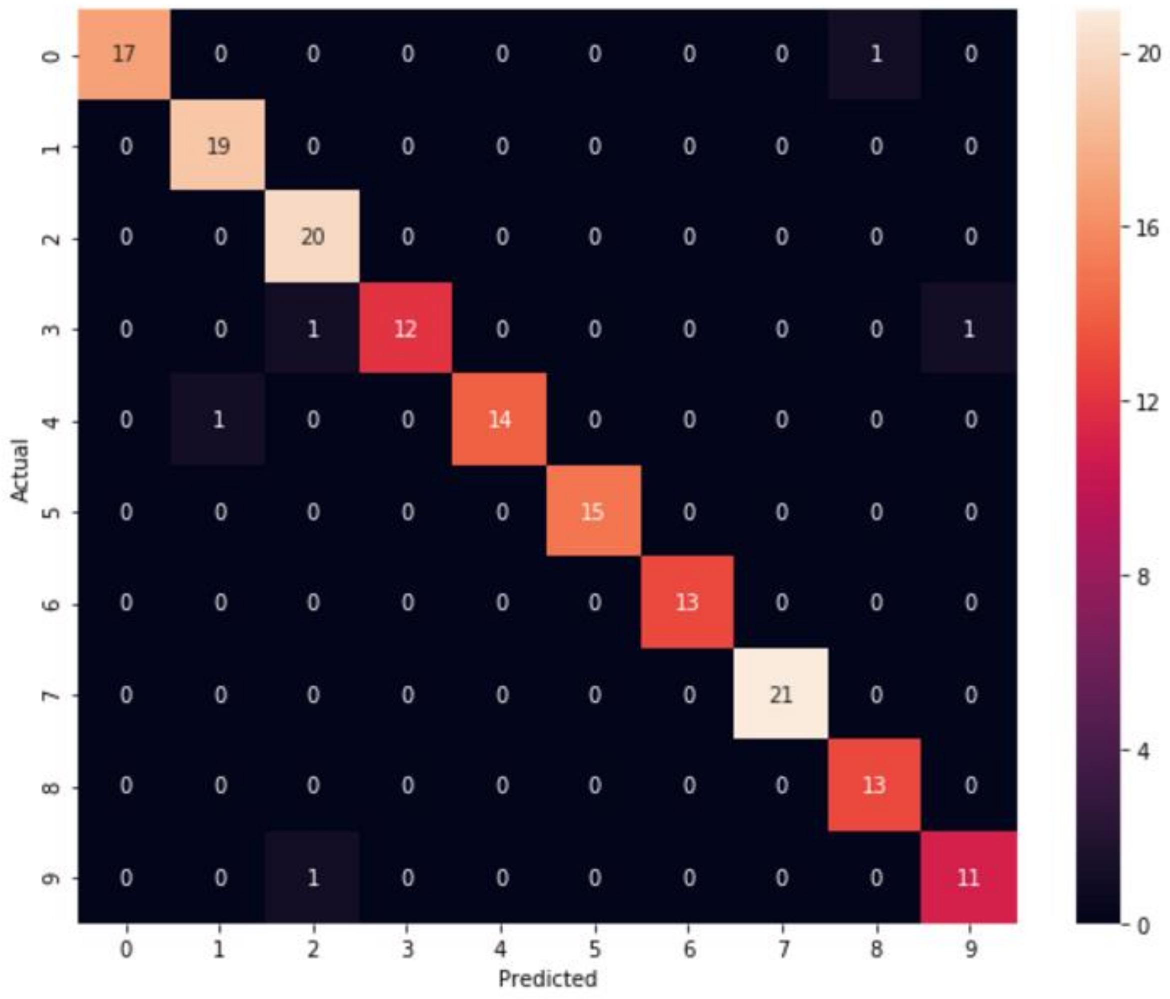
Model Evaluation and Validation

Final results can be seen below,

	Score	Score (after apply PCA)
SVM ("Linear Kernel")	0.95625	0.96875
SVM("rbf" kernel)	0.925	0.9625
SVM("poly" kernel)	0.86875	0.96875
SVM("sigmoid" kernel)	0.9125	0.9125

The above values clearly shows that this is good candidate for classification problem.

PCA has improvement on Linear, rbf and poly (degree=2) however no effect on sigmoid (degree=2). Linear Kernel is good fit for the solution of this problem. It’s confusion matrix as given below,

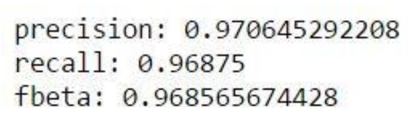


The "Actual" axis indicates the actual numbers, and the "Predicted" indicates their predicted results, for all rows in the test data set. In the middle of the diagonal line, numbers are displayed where the actual figures were predicted correctly, and the remaining columns indicate errors.

There are total 5 errors where,

- 0 predicted as 8
- 3 Predicted as 2 and 9
- 4 predicted as 1
- 9 predicted as 2

Precision, Recall and F beta score is given below,



```
precision: 0.970645292208
recall: 0.96875
fbeta: 0.968565674428
```

Where precision and recall is defined below,

recall = True positive / (True positive + False negative) precision = True positive / (True positive + False positive)

F score is harmonic mean of precision and recall.

When we checked F score 0.968 is pretty good.

On the other hand, the model is tested for several times with different kind of splits with sklearn train_test_split function. But it is not tested with any other dataset, because the only data set that I have found is Minst dataset which is not allowed for this project.

Justification

The final model has an accuracy of 96.87%, which is higher than 95.62%. It is also worthwhile to evaluate and compare models more subjectively. In particular, the reference model used just SVMA in part for its simplicity. In this respect, final model is also advantageous differs.

In addition, our model was also chosen for its performance on precision and recall that works well. In general, I think that the final model is a good model that helps to solve the problem. From possibility to more accurately predict digits, as well as a model that allows.

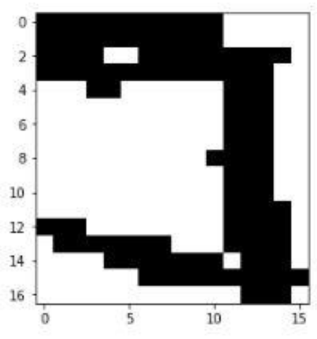
V. Conclusion

Free-Form Visualization

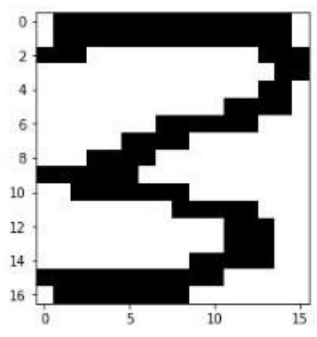
As seen from confusion matrix there are 5 false predictions, which are,

- 0 predicted as 8
- 3 Predicted as 2 and 9
- 4 predicted as 1
- 9 predicted as 2

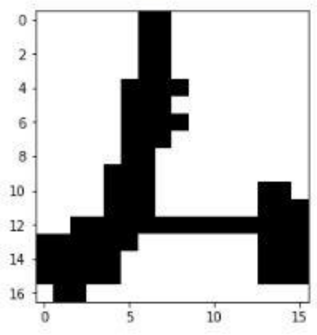
It can be visually inspected below,



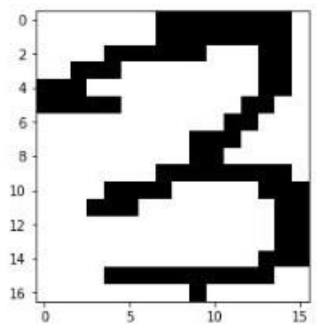
```
dig            9
IMG_test_pca    2
Name: 1172, dtype: int64
```



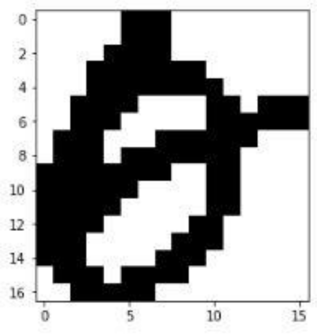
```
dig            3
IMG_test_pca    2
Name: 469, dtype: int64
```



```
dig            4
IMG_test_pca    1
Name: 682, dtype: int64
```



```
dig            3
IMG_test_pca    9
Name: 935, dtype: int64
```



```
dig            0
IMG_test_pca    8
Name: 1196, dtype: int64
```

Reflection

Recognition of the character is fundamental, but the most difficult in the field of pattern recognition with a large number of useful applications. This was an intensive field of research, since because it's a natural way interactions between computers and people. I did a project to classify handwritten figures using machine learning in Python. I used Python Scikit Learn library together with Pandas and Numpy in an open data set for reading and classifying handwritten digits.

A solution of the SVM style was given to the analysis of the PCA. Conceptually, the formulation considers the problem as a one-class modeling problem with zero target value around which one maximizes the variance. However, as it is handwritten it is not easy to classify properly. This is the main difficuilty to solve this kind of problem with classification aspect.

However, even handwritten digits seen confused during eye inspection, the algorithm very well classify the labels. This solution is fit my expectations moreover beyond it.

Improvement

CNN would be better candidate for solution. However, in my point of view Support Vector Machines often exceeds CNN, because they avoid the two main disadvantages of CNN:

1- CNN often converge on local minima, rather than in global minima, which means that they essentially "skip a big picture" sometimes

2- ANNs are often overfit if training takes too long, which means that for any given pattern, the ANN can begin to treat noise as part of the template.

On the other hand, in order to classify handwritten figures, ten vector support machines is needed. Each vector support machine recognizes exactly one digit and does not recognize all the others. Since each handwritten figure can not contain more information than just its class, there is no point in trying to solve this problem using a CNN where have a bunch of hidden layers with sizes from h1 to hn depending on the number of functions, as well as on the offset parameters, and they make up your model. SVM consists of a set of supporting vectors selected from a set of workouts, with a weight for each. In the worst case, the number of reference vectors is exactly the number of training samples (although this mainly occurs with small sets of exercises or in degenerate cases), and in general its model size weighs linearly.

As a result, solution selection is an optimization problem itself. For this particular project I choose SVM with PCA as solution however lost at least 2% better result. If CNN is choosed I probably get 2-3% better results but needed GPU power and more time to test each step.