Random Forest, PCA and KNN-MNIST dataset

Siyuan Zhang

1 Resources

Student: Siyuan Zhang
Language: Python
Code: https://github.com/Siyuan-gwu/Compare-Random_Forest-PCA-KNN-MNIST-dataset
Instruction: The instruction to run the code is on github (readme)
Resource:

1. https://www.kaggle.com/sflender/comparing-random-forest-pca-and-knn
2. https://iq.opengenus.org/algorithm-principal-component-analysis-pca/

```
import numpy as np
import pandas as pd
import seaborn as sb
import os
import matplotlib.pyplot as plt, matplotlib.image as mpimg

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
import pylab
```

2 Dataset details

There are two datasets, training data and testing data

1. The training data contains 28000 images
2. Each image has 784 pixels in total, each single pixel-value indicates the lightness or darkness of that pixel
3. The first column is the actual digit of the image
4. The testing data is the same with training data, but without “label” column to represent the actual number

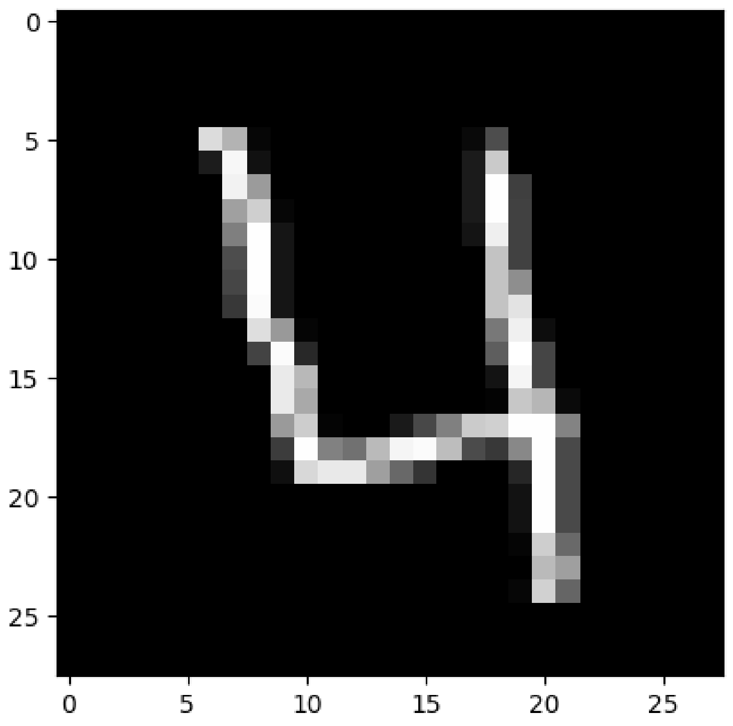
Inspect the Dataset

```
#read the data
train = pd.read_csv('train.csv')
print (train.head(5))
```

```
   label  pixel0  pixel1  pixel2  ...  pixel780  pixel781  pixel782  pixel783
0      1       0       0       0  ...         0         0         0         0
1      0       0       0       0  ...         0         0         0         0
2      1       0       0       0  ...         0         0         0         0
3      4       0       0       0  ...         0         0         0         0
4      0       0       0       0  ...         0         0         0         0

[5 rows x 785 columns]
```

Print an example of the number

```
#plot a number "4"
img = train.values[3]
img = img.reshape(28,28)
plt.imshow(img, cmap='gray')
plt.show()
```



Data pre-processing

Drop the label of train dataset.

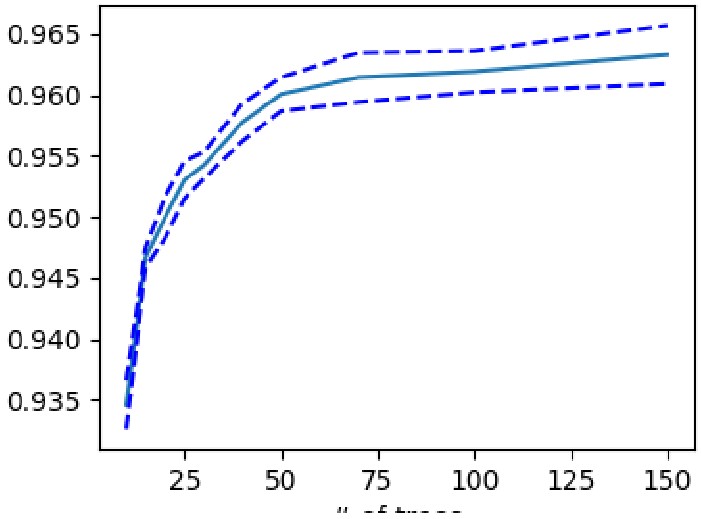
```
train = train.drop("label", 1)
```

Do the data normalization, if the pixel larger than one, then change it to one. Otherwise, do nothing.

Check the performance of random forest classifier

```
def random_forest():
    # loading training data
    print('Loading training data')
    X_tr = train.values[:, 1:].astype(float)
    y_tr = train.values[:, 0]

    scores = list()
    scores_std = list()

    print('Start learning...')
    n_trees = [10, 15, 20, 25, 30, 40, 50, 70, 100, 150]
    for n_tree in n_trees:
        print(n_tree)
        recognizer = RandomForestClassifier(n_tree)
        score = cross_val_score(recognizer, X_tr, y_tr)
        scores.append(np.mean(score))
        scores_std.append(np.std(score))

    sc_array = np.array(scores)
    std_array = np.array(scores_std)
    print('Score: ', sc_array)
    print('Std  : ', std_array)

    plt.figure(figsize=(4,3))
    plt.plot(n_trees, scores)
    plt.plot(n_trees, sc_array + std_array, 'b--')
    plt.plot(n_trees, sc_array - std_array, 'b--')
    plt.ylabel('CV score')
    plt.xlabel('# of trees')
    plt.savefig('cv_trees.png')
    plt.show()
```

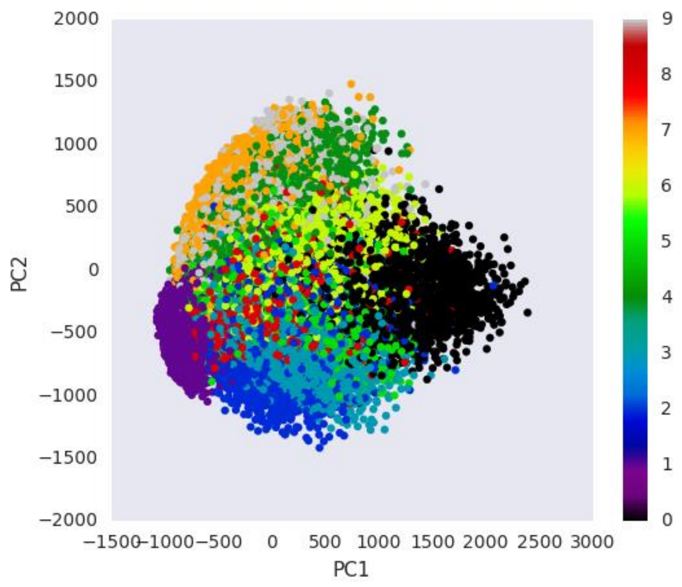
Result:



The x-axis is the number of trees
The y-axis is the rate of accuracy

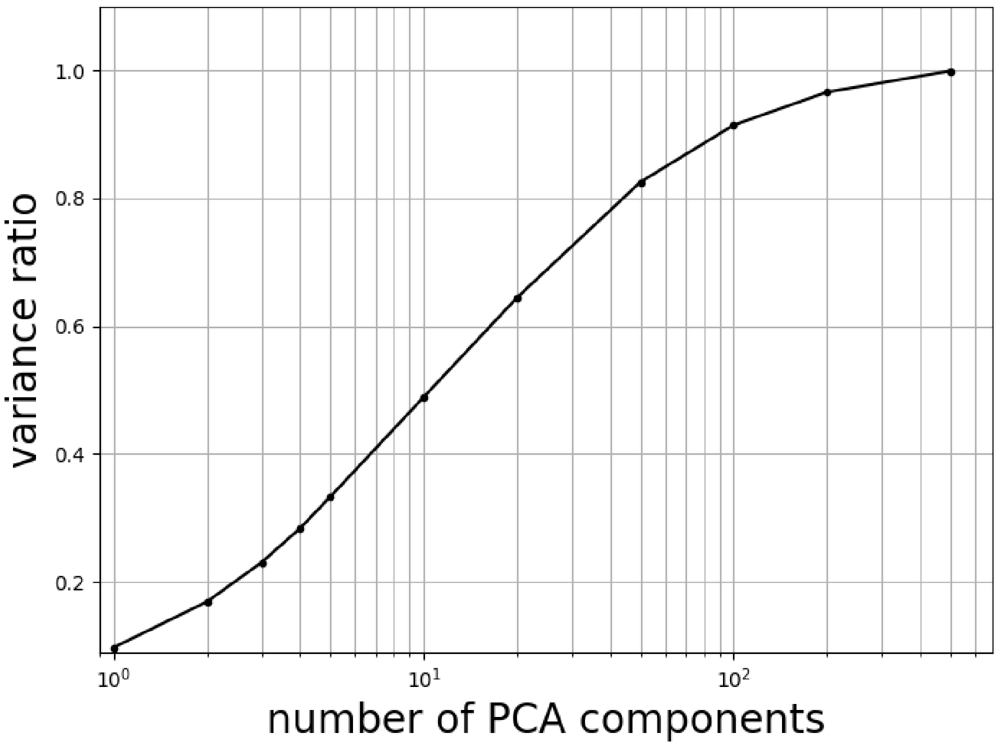
Try on PCA and KNN (we use component = 2)

```
#try on PCA
pca = PCA(n_components=2)
pca.fit(train)
transform = pca.transform(train)


plt.figure()
plt.scatter(transform[:,0],transform[:,1], s=20, c = target, cmap = "nipy_spectral", edgecolor = "None")
plt.colorbar()
plt.clim(0,9)


plt.xlabel("PC1")
plt.ylabel("PC2")
```



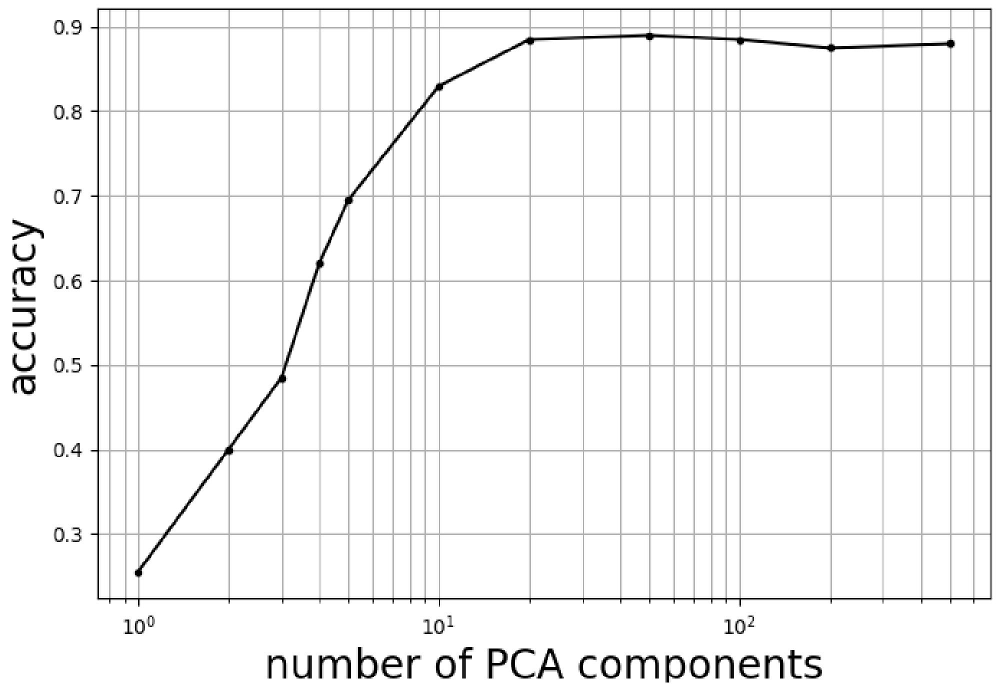
We notice that PCA separates the feature space into visible clusters already for 2 components. Next, I am going to check how many components are needed to capture most of the variance in the data.

```
n_components_array=([1,2,3,4,5,10,20,50,100,200,500])
vr = np.zeros(len(n_components_array))
i=0;
for n_components in n_components_array:
    pca = PCA(n_components=n_components)
    pca.fit(train)
    vr[i] = sum(pca.explained_variance_ratio_)
    i=i+1
plt.figure(figsize=(8,4))
plt.plot(n_components_array,vr,'k.-')
plt.xscale("log")
plt.ylim(9e-2,1.1)
plt.xlim(0.9)
plt.grid(which="both")
plt.xlabel("number of PCA components",size=20)
plt.ylabel("variance ratio",size=20)
plt.show()
```



We notice that ~100 PCA components are needed to capture ~90% of the variance in the data. But I cannot know the prediction result from this plot. Next, I would train a KNN classifier on the PCA output.

```
clf = KNeighborsClassifier()
n_components_array=([1,2,3,4,5,10,20,50,100,200,500])
score_array = np.zeros(len(n_components_array))
i=0
for n_components in n_components_array:
    pca = PCA(n_components=n_components)
    pca.fit(train)
    transform = pca.transform(train.iloc[0:1000])
    score_array[i] = evaluate_classifier(clf, transform, target.iloc[0:1000], 0.8)
    i=i+1
plt.figure(figsize=(8,4))
plt.plot(n_components_array,score_array,'k.-')
plt.xscale('log')
plt.xlabel("number of PCA components", size=20)
plt.ylabel("accuracy", size=20)
plt.grid(which="both")
plt.show()
```



We notice that the accuracy seems to saturate at ~90% for >~20 PCA components. In fact, the accuracy even seems to drop for much larger numbers, even though a larger number of PCA components captures more of the variance in the data. Maybe the drop of the accuracy is probably due to overfitting.

Result

Submit the knn+PCA and Random Forest test result on Kaggle

```
# random forest classification test result
clf = RandomForestClassifier(n_estimators = 100, n_jobs=1, criterion="gini")
clf.fit(train, target)
results=clf.predict(test)
np.savetxt('result_rf.csv',
        np.c_[range(1,len(test)+1),results],
        delimiter=',',
        header = 'ImageId,Label',
        comments = '',
        fmt='%d')
```

Get the result score: 0.95

```
# PCA and KNN test result
pca = PCA(n_components=50)
pca.fit(train)
transform_train = pca.transform(train)
transform_test = pca.transform(test)

clf = KNeighborsClassifier()
clf.fit(transform_train, target)
results = clf.predict(transform_test)
np.savetxt('result_knn_pca.csv',
        np.c_[range(1, len(test) + 1), results],
        delimiter=',',
        header='ImageId,Label',
        comments='',
        fmt='%d')
```

Get the result score: 0.964
It seems that the PCA and KNN accuracy is higher.

3 Algorithm Description

PCA algorithm

Principal component analysis (PCA) is a technique to bring out strong patterns in a dataset by suppressing variations. It is used to clean data sets to make it easy to explore and analyze. The algorithm of Principal Component Analysis is based on a few mathematical ideas namely:

1. Variance and Covariance
2. Eigen Vectors and Eigen values

Algorithm steps:
1. Get your data
2. Give your data a structure
3. Standardize your data
4. Get covariance of matrix Z
5. Calculate Eigen Vectors and Eigen Values
6. Sort the Eigen Vectors

7. Calculate the new features
8. Drop unimportant features from the set

4 Runtime

For PCA:

Suppose we have n data points, and each represented with p features.
Covariances matrix computation is $O(p^2 * n)$, its eigen-value decomposition is $O(p\text{^}3)$.
So, the time complexity of PCA is $O(p^2 * n + p\text{^}3)$