LAB9 all Parts

April 21, 2021

1. Review the attached papers (that's a chapter from the textbook but could not print it as one pdf)
2. Find a previous lab that uses the iris dataset
3. Tasks for the lab: 1 - complete pca as attached then 2 - complete the same task as in the previous lab using the reduced features (1 instead of 3)

0.0.1 Part1: Understanding of PCA

0.0.2 Reference Website for part 1 work

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

[1]:
```
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# load dataset into Pandas DataFrame
df = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal␣
 ↪width','target'])
```

[2]:
```
df.head(3)
```

[2]:
```
   sepal length  sepal width  petal length  petal width       target
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
```

Standardize the Data PCA is effected by scale so you need to scale the features in your data before applying PCA. Use StandardScaler to help you standardize the dataset's features onto unit scale (mean = 0 and variance = 1) which is a requirement for the optimal performance of many machine learning algorithms. If you want to see the negative effect not scaling your data can have, scikit-learn has a section on the effects of not standardizing your data.

[3]:
```
from sklearn.preprocessing import StandardScaler
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
# Separating out the features
x = df.loc[:, features].values
# Separating out the target
y = df.loc[:,['target']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

PCA Projection to 2D The original data has 4 columns (sepal length, sepal width, petal length, and petal width). In this section, the code projects the original data which is 4 dimensional into 2 dimensions. I should note that after dimensionality reduction, there usually isn't a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation.

[4]:

```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
```

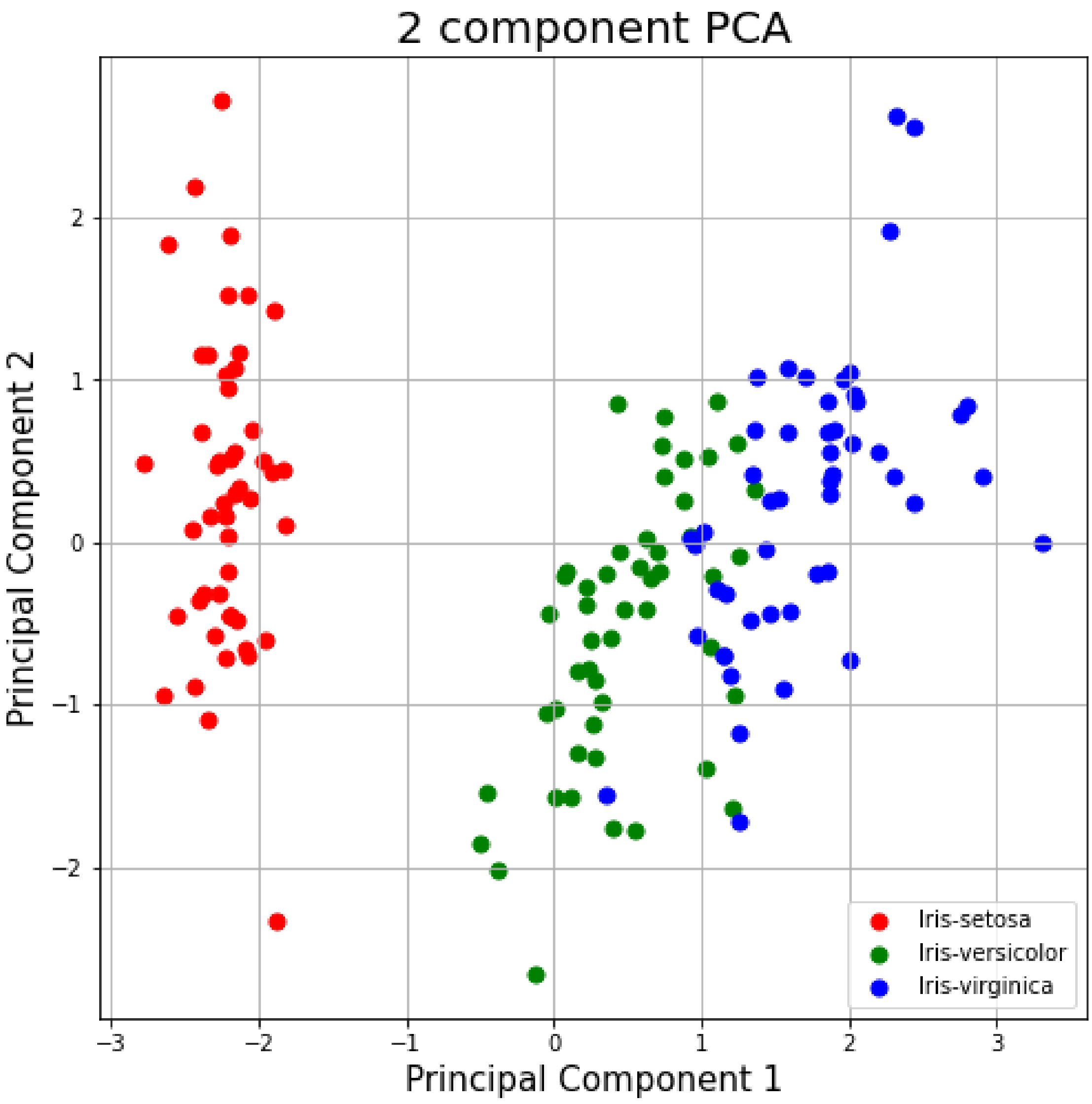
[5]:

```python
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
```

[6]:

```python
from matplotlib import pyplot as plt

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



0.0.3 Explained Variance

The explained variance tells you how much information (variance) can be attributed to each of the principal components. This is important as while you can convert 4 dimensional space to 2 dimensional space, you lose some of the variance (information) when you do this. By using the attribute explained_variance_ratio_, you can see that the first principal component contains 72.77% of the variance and the second principal component contains 23.03% of the variance. Together, the two components contain 95.80% of the information.

[7]:

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

[8]:

```
from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set
train_img, test_img, train_lbl, test_lbl = train_test_split( mnist.data, mnist.
 ↪target, test_size=1/7.0, random_state=0)
```

[9]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(train_img)
# Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

[10]:

```
from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.95)
```

[11]:

```
pca.fit(train_img)
```

[11]:

```
PCA(n_components=0.95)
```

[12]:

```
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

[13]:

```
from sklearn.linear_model import LogisticRegression
# all parameters not specified are set to their defaults
# default solver is incredibly slow which is why it was changed to 'lbfgs'
logisticRegr = LogisticRegression(solver = 'lbfgs')
```

[14]:

```
logisticRegr.fit(train_img, train_lbl)
```

```
/Users/hidenaka/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[14]:

```
LogisticRegression()
```

[15]:

```
# Predict for One Observation (image)
logisticRegr.predict(test_img[0].reshape(1,-1))
```

[15]:
```
array(['0'], dtype=object)
```

[16]:
```
# Predict for One Observation (image)
logisticRegr.predict(test_img[0:10])
```

[16]:
```
array(['0', '4', '1', '2', '4', '7', '7', '1', '1', '7'], dtype=object)
```

[17]:
```
logisticRegr.score(test_img, test_lbl)
```

[17]:
```
0.9201
```

0.0.4 Part2: Without PCA and With PCA Performance models using diabetes datasets

[18]:
```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

[19]:
```
# Task: Load Data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('diabetes.csv')
df.head(3)
```

[19]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

[20]:
```
# Task: Data Spliting
X = df.drop(['Outcome'],axis =1)
y = df.Outcome.values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
```

[21]:
```python
import numpy as np
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
```

0.0.5 KNN without PCA

[22]:
```python
# Import module for KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)

# Fit (i.e. traing) the model
knn.fit(X_train, y_train)

# Use the .predict() method to make predictions from the X_test subset
pred = knn.predict(X_test)

# Import classification report and confusion matrix to evaluate predictions
from sklearn.metrics import classification_report, confusion_matrix

# Print out classification report and confusion matrix
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.78      0.78       157
           1       0.52      0.50      0.51        74

    accuracy                           0.69       231
   macro avg       0.64      0.64      0.64       231
weighted avg       0.69      0.69      0.69       231
```

0.0.6 KNN with PCA

[23]:
```python
# Use 1 Principle component to train algorithm

from sklearn.decomposition import PCA

pca = PCA(n_components=0.95)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

[24]:
```python
from sklearn.ensemble import RandomForestClassifier

knnpca = knn.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = knn.predict(X_test)
```

[25]:

```
# Performance Evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy with using one principle component  : ",accuracy_score(y_test,␣
 ↪y_pred))
```

```
[[137  20]
 [ 37  37]]
Accuracy with using one principle component  :  0.7532467532467533
```

Using KNN:
From the above experimentation we achieved optimal level of accuracy while significantly reducing the number of features in the dataset. We saw that accuracy achieved with only 1 principal component is higher than the accuracy achieved with will feature set.

From the results we can see that the accuracy achieved with one principal component (75.32%) was greater than the one achieved with all features (69.0%).

[26]:

```
#from sklearn.decomposition import PCA

#pca = PCA(n_components=1)
#X_train = pca.fit_transform(X_train)
#X_test = pca.transform(X_test)
```

[27]:

```
# Use 2 and 3 Principle component to train algorithm

#pca = PCA(n_components=0)
#X_train = pca.fit_transform(X_train)
#X_test = pca.transform(X_test)
#knnpca = knn.fit(X_train, y_train)

# Predicting the Test set results
#y_pred = knn.predict(X_test)

#cm = confusion_matrix(y_test, y_pred)
#print(cm)
#print("Accuracy with using one principle component  : ",accuracy_score(y_test,␣
 ↪y_pred))

# can't do with this datasets
```

PCA should be used mainly for variables which are strongly correlated. If the relationship is weak between variables, PCA does not work well to reduce data. Refer to the correlation matrix to determine. In general, if most of the correlation coefficients are smaller than 0.3, PCA will not help.

[ ]:

[ ]: