Feature Dimention Reduction Using LDA and PCA with Python I Principal Component Analysis in Feature Selection

What is LDA (Linear Discriminant Analysis)?

The idea behind LDA is simple. Mathematically speaking, we need to find a new feature space to project the data in order to maximize classes separability

Linear Discriminant Analysis is a supervised algorithm as it takes the class label into consideration. It is a way to reduce ‘dimensionality’ while at the same time preserving as much of the class discrimination information as possible.

LDA helps you find the boundaries around clusters of classes. It projects your data points on a line so that your clusters are as separated as possible, with each cluster having a relative (close) distance to a centroid.

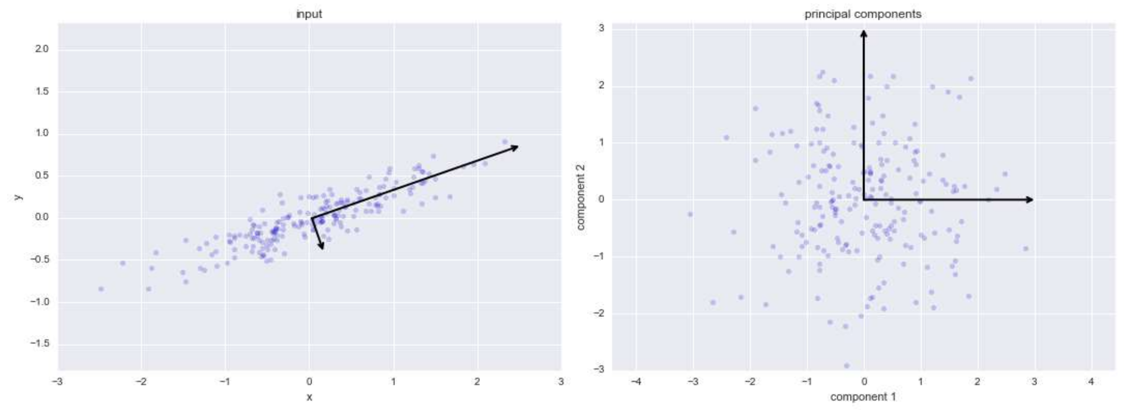
So the question arises- how are these clusters are defined and how do we get the reduced feature set in case of LDA?

Basically LDA finds a centroid of each class datapoints. For example with thirteen different features LDA will find the centroid of each of its class using the thirteen different feature dataset. Now on the basis of this, it determines a new dimension which is nothing but an axis which should satisfy two criteria: 1. Maximize the distance between the centroid of each class. 2. Minimize the variation (which LDA calls scatter and is represented by s2), within each category.

What is PCA

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation.

Dimensions are nothing but features that represent the data. For example, A 28 X 28 image has 784 picture elements (pixels) that are the dimensions or features which together represent that image.



One important thing to note about PCA is that it is an Unsupervised dimensionality reduction technique, you can cluster the similar data points based on the feature correlation between them without any supervision (or labels), and you will learn how to achieve this practically using Python in later sections of this tutorial!

According to Wikipedia, PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

When to use PCA

Data Visualization: When working on any data related problem, the challenge in today's world is the sheer volume of data, and the variables/features that define that data. To solve a problem where data is the key, you need extensive data exploration like finding out how the variables are correlated or understanding the distribution of a few variables. Considering that there are a large number of variables or dimensions along which the data is distributed, visualization can be a challenge and almost impossible.

Speeding Machine Learning (ML) Algorithm: Since PCA's main idea is dimensionality reduction, you can leverage that to speed up your machine learning algorithm's training and testing time considering your data has a lot of features, and the ML algorithm's learning is too slow.

How to do PCA

We can calculate a Principal Component Analysis on a dataset using the PCA() class in the scikit-learn library. The benefit of this approach is that once the projection is calculated, it can be applied to new data again and again quite easily.

When creating the class, the number of components can be specified as a parameter.

The class is first fit on a dataset by calling the fit() function, and then the original dataset or other data can be projected into a subspace with the chosen number of dimensions by calling the transform() function.

Once fit, the eigenvalues and principal components can be accessed on the PCA class via the explained_variance_ and components_ attributes.

In [27]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [28]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
```

In [29]:
```
data = pd.read_csv('Subject01_dataset_202002071_50ms.csv', nrows = 2861)
data.dropna(inplace=True)
data.head()
```

Out[29]:

	Time	DEMG1_AR1	DEMG1_AR2	DEMG1_AR3	DEMG1_AR4	DEMG1_AR5	DEMG1_AR6	DEM
0	0.00	0.622484	-0.260461	0.356585	-0.234134	0.211762	-0.032847	
1	0.05	0.104438	0.112400	0.282052	-0.097667	0.134183	-0.088160	
2	0.10	0.032687	0.044736	0.055276	0.095904	0.082075	-0.239689	
3	0.15	0.148638	0.044486	0.156377	-0.141641	-0.132405	0.049945	
4	0.20	0.298830	-0.126001	0.336519	-0.123507	-0.132748	-0.037417	

5 rows × 103 columns

In [30]:
```
data = pd.read_csv('Subject01_dataset_202002071_50ms.csv', nrows = 2861)
data.dropna(inplace=True)
data.head()
```

Out[30]:

	Time	DEMG1_AR1	DEMG1_AR2	DEMG1_AR3	DEMG1_AR4	DEMG1_AR5	DEMG1_AR6	DEM
0	0.00	0.622484	-0.260461	0.356585	-0.234134	0.211762	-0.032847	
1	0.05	0.104438	0.112400	0.282052	-0.097667	0.134183	-0.088160	
2	0.10	0.032687	0.044736	0.055276	0.095904	0.082075	-0.239689	
3	0.15	0.148638	0.044486	0.156377	-0.141641	-0.132405	0.049945	
4	0.20	0.298830	-0.126001	0.336519	-0.123507	-0.132748	-0.037417	

5 rows × 103 columns

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
```

Remove Constant, Quasi Constant and Duplicate Features

In [32]:
```
#remove constant and quasi constant features
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(X_train)
X_train_filter = constant_filter.transform(X_train)
X_test_filter = constant_filter.transform(X_test)
```

In [33]:
```
X_train_filter.shape, X_test_filter.shape
```

Out[33]:
```
((25148, 6), (6288, 6))
```

In [34]:
```
#remove duplicate features
X_train_T = X_train_filter.T
X_test_T = X_test_filter.T
```

In [35]:
```
X_train_T = pd.DataFrame(X_train_T)
X_test_T = pd.DataFrame(X_test_T)
```

In [36]:
```
X_train_T.duplicated().sum()
```

Out[36]:
```
0
```

In [37]:
```
duplicated_features = X_train_T.duplicated()
```

In [38]:
```
features_to_keep = [not index for index in duplicated_features]

X_train_unique = X_train_T[features_to_keep].T
X_test_unique = X_test_T[features_to_keep].T
```

In [39]:
```
scaler = StandardScaler().fit(X_train_unique)
X_train_unique = scaler.transform(X_train_unique)
X_test_unique = scaler.transform(X_test_unique)
```

In [40]:
```
X_train_unique = pd.DataFrame(X_train_unique)
X_test_unique = pd.DataFrame(X_test_unique)
```

In [41]:
```
X_train_unique.shape, X_test_unique.shape
```

Out[41]:
```
((25148, 6), (6288, 6))
```

Removal of correlated Feature

In [42]:
```
corrmat = X_train_unique.corr()
```

In [43]:
```
#find correlated features
def get_correlation(data, threshold):
    corr_col = set()
    corrmat = data.corr()
    for i in range(len(corrmat.columns)):
        for j in range(i):
            if abs(corrmat.iloc[i, j]) > threshold:
                colname = corrmat.columns[i]
                corr_col.add(colname)
    return corr_col

corr_features = get_correlation(X_train_unique, 0.70)
print('correlated features: ', len(set(corr_features)) )
```

```
correlated features:  0
```

In [44]:
```
X_train_uncorr = X_train_unique.drop(labels=corr_features, axis = 1)
X_test_uncorr = X_test_unique.drop(labels = corr_features, axis = 1)
```

In [45]:
```
X_train_uncorr.shape, X_test_uncorr.shape
```

Out[45]:
```
((25148, 6), (6288, 6))
```

In [ ]:

Feature Dimention Reduction by LDA or Is it a Classifier

In [20]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

In [21]:
```
lda = LDA(n_components=1)
X_train_lda = lda.fit_transform(X_train_uncorr, y_train)
X_test_lda = lda.transform(X_test_uncorr)
```

In [22]:
```
X_train_lda.shape, X_test_lda.shape
```

Out[22]:
```
((25148, 1), (6288, 1))
```

In [23]:
```
def run_randomForest(X_train, X_test, y_train, y_test):
    clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy on test set: ')
    print(accuracy_score(y_test, y_pred))
```

In [24]:
```
%%time
run_randomForest(X_train_lda, X_test_lda, y_train, y_test)
```

```
Accuracy on test set:
0.3821564885496183
CPU times: total: 26.8 s
Wall time: 5.39 s
```

In [25]:
```
%%time
run_randomForest(X_train_uncorr, X_test_uncorr, y_train, y_test)
```

```
Accuracy on test set:
0.9209605597964376
CPU times: total: 29 s
Wall time: 5.69 s
```

In [26]:
```
%%time
run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy on test set:
0.9212786259541985
CPU times: total: 32 s
Wall time: 6.59 s
```

Feature Reduction by PCA?

In [46]:
```
from sklearn.decomposition import PCA
```

In [47]:
```
pca = PCA(n_components=2, random_state=42)
pca.fit(X_train_uncorr)
```

Out[47]:
```
▾                PCA
PCA(n_components=2, random_state=42)
```

In [48]:
```
X_train_pca = pca.transform(X_train_uncorr)
X_test_pca = pca.transform(X_test_uncorr)
X_train_pca.shape, X_test_pca.shape
```

Out[48]:
```
((25148, 2), (6288, 2))
```

In [49]:
```
%%time
run_randomForest(X_train_pca, X_test_pca, y_train, y_test)
```

```
Accuracy on test set:
0.5221055979643766
CPU times: total: 23.2 s
Wall time: 4.91 s
```

In [50]:
```
%%time
run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy on test set:
0.9212786259541985
CPU times: total: 26.3 s
Wall time: 5.29 s
```

In [51]:
```
X_train_uncorr.shape
```

Out[51]:
```
(25148, 6)
```

In [52]:
```
for component in range(1,30):
    pca = PCA(n_components=component, random_state=42)
    pca.fit(X_train_uncorr)
    X_train_pca = pca.transform(X_train_uncorr)
    X_test_pca = pca.transform(X_test_uncorr)
    print('Selected Components: ', component)
    run_randomForest(X_train_pca, X_test_pca, y_train, y_test)
    print()
```

```
Selected Components:  1
Accuracy on test set:
0.29182569974554706

Selected Components:  2
Accuracy on test set:
0.5221055979643766

Selected Components:  3
Accuracy on test set:
0.6983142493638677

Selected Components:  4
Accuracy on test set:
0.8121819338422391

Selected Components:  5
Accuracy on test set:
0.862118320610687

Selected Components:  6
Accuracy on test set:
0.8991730279898219
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[52], line 3
      1 for component in range(1,30):
      2     pca = PCA(n_components=component, random_state=42)
----> 3     pca.fit(X_train_uncorr)
      4     X_train_pca = pca.transform(X_train_uncorr)
      5     X_test_pca = pca.transform(X_test_uncorr)

File ~\AppData\Roaming\Python\Python38\site-packages\sklearn\decomposition\_p
ca.py:435, in PCA.fit(self, X, y)
    417 """Fit the model with X.
    418 
    419 Parameters
   (...)
    431     Returns the instance itself.
    432 """
    433 self._validate_params()
--> 435 self._fit(X)
    436 return self

File ~\AppData\Roaming\Python\Python38\site-packages\sklearn\decomposition\_p
ca.py:512, in PCA._fit(self, X)
    510 # Call different fits for either full or truncated SVD
    511 if self._fit_svd_solver == "full":
--> 512     return self._fit_full(X, n_components)
    513 elif self._fit_svd_solver in ["arpack", "randomized"]:
    514     return self._fit_truncated(X, n_components, self._fit_svd_solver)

File ~\AppData\Roaming\Python\Python38\site-packages\sklearn\decomposition\_p
ca.py:526, in PCA._fit_full(self, X, n_components)
    522         raise ValueError(
    523             "n_components='mle' is only supported if n_samples >= n_f
eatures"
    524         )
    525 elif not 0 <= n_components <= min(n_samples, n_features):
--> 526     raise ValueError(
    527         "n_components=%r must be between 0 and "
    528         "min(n_samples, n_features)=%r with "
    529         "svd_solver='full'" % (n_components, min(n_samples, n_feature
s))
    530     )
    532 # Center data
    533 self.mean_ = np.mean(X, axis=0)

ValueError: n_components=7 must be between 0 and min(n_samples, n_features)=6
with svd_solver='full'
```

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```