VIVIAN JOSEPH

(20BCE0777)

Machine Learning

Lab DA-5

Dimensionality Reduction and Unbalanced Data Processing

PCA:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
import seaborn as sn
from sklearn.preprocessing import StandardScaler
```

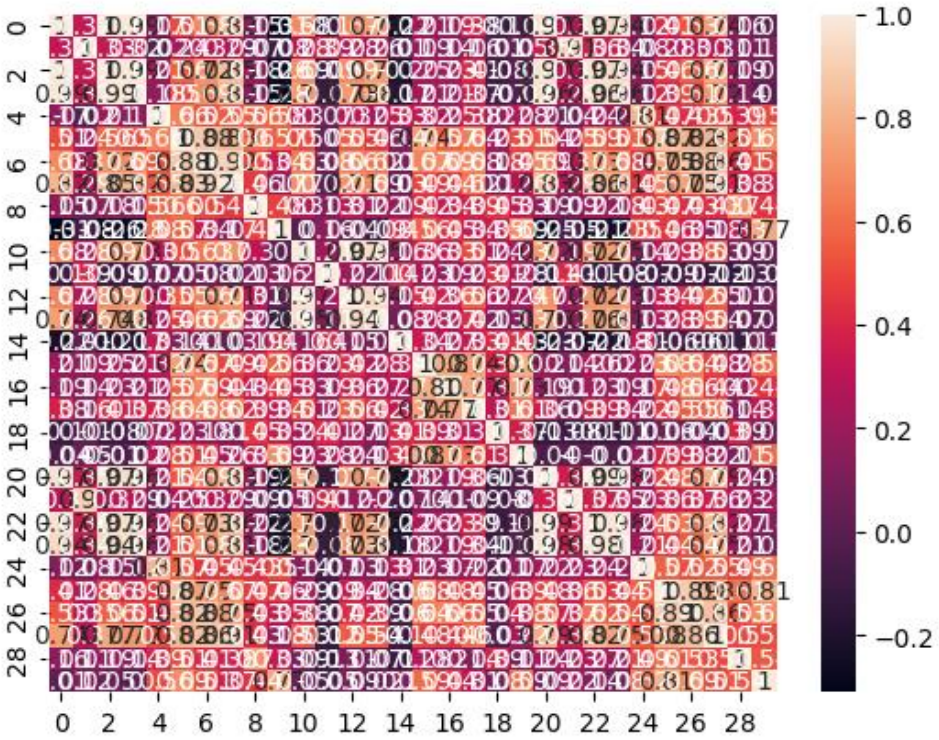
Load the breast cancer dataset:

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X = cancer.data
X=pd.DataFrame(X)
y = cancer.target
```

Visualizing the correlations between features

```
correlation_matrix = X.corr().round(2)
# annot = True to print the values inside the square
sn.heatmap(data=correlation_matrix, annot=True)
```



Total 30 input features (Clearly, too many features)

Training and testing without PCA:

```
X_train, X_test, Y_train, Y_test = train_test_split( X, y, test_size=0.30,
random_state=30)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lm = LinearRegression()
lm.fit(X_train, Y_train)
# model evaluation for training set
y_train_predict = lm.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 0.22817108093507243
R2 score is 0.7783108675080304
```

```
# model evaluation for testing set
Y_pred = lm.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_pred)))
r2 = r2_score(Y_test, Y_pred)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

# model evaluation for testing set
print(Y_pred.shape)
Y_pred
pd.DataFrame(Y_pred,Y_test)
```

```
The model performance for testing set
--------------------------------------
RMSE is 0.2471775150359638
R2 score is 0.735642304718755
```

Test R2 Score with 30 features

Applying PCA:

```
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Initialize PCA model
pca = PCA(n_components=4)

# Fit and transform data
X_pca = pca.fit_transform(X_std)

# Plot the transformed data
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y+1)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Iris dataset after PCA')
plt.show()

# Print explained variance ratio
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

final shape:

```
X_pca.shape
✓ 0.1s
(569, 4)
```

Training and testing after PCA:

```
X_train, X_test, Y_train, Y_test = train_test_split( X_pca, y, test_size=0.30,
random_state=30)
```

```
lm = LinearRegression()
lm.fit(X_train, Y_train)
# model evaluation for training set
y_train_predict = lm.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 0.2708734948782267
R2 score is 0.6875674837138712
```

```
Y_pred = lm.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_pred)))
r2 = r2_score(Y_test, Y_pred)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

# model evaluation for testing set
print(Y_pred.shape)
Y_pred

pd.DataFrame(Y_pred,Y_test)
```

```
The model performance for testing set
--------------------------------------
RMSE is 0.25777807147461257
R2 score is 0.7124813809428769
```

Test R2 Score with only 4 features

Number of input Features	R2 Score	RMSE
30 input features	0.735	0.247
only **6** input features	0.712	0.258

Clearly, the r2 score is affected barely even after drastically reducing the number of features from 30 to 6.

Under-Sampling and Over-Sampling:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import r2_score
import seaborn as sn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Creating an imbalanced dataset:

```
from sklearn.datasets import load_breast_cancer
import numpy as np

breast_cancer = load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target

# Generate an imbalanced version of the dataset with a 90/10 split
np.random.seed(42)
mask = np.random.rand(len(y)) < 0.9 # 90% of samples in one class
y[mask] = 0 # Assign the first class to the majority of samples
y[~mask] = 1 # Assign the second class to the minority of samples

print("Original class distribution:", np.bincount(breast_cancer.target))
print("Imbalanced class distribution:", np.bincount(y))
```

target data is highly imbalanced

```
Original class distribution: [504  65]
Imbalanced class distribution: [504  65]
```

(504 samples with label 0, 65 samples with label 1)

Over sample the data:

```
# Get the indices of the majority and minority classes
majority_class = np.where(y == 0)[0]
minority_class = np.where(y == 1)[0]
# Randomly select a subset of samples from the minority class
random_indices = np.random.choice(minority_class, size=len(majority_class), replace=True)

# Combine the majority class samples with the randomly duplicated minority class samples
oversampled_indices = np.concatenate([majority_class, random_indices])
X_oversampled = X[oversampled_indices]
y_oversampled = y[oversampled_indices]

# Check the class distribution
print(np.unique(y_oversampled, return_counts=True))
```

Oversampled Data:

```
(array([0, 1]), array([504, 504], dtype=int64))
```

(504 samples with label 0, 504 samples with label 1)

Under sample the data:

```
num_class0 = np.sum(y == 0)
num_class1 = np.sum(y == 1)

# Randomly select a subset of samples from the minority class
random_indices = np.random.choice(minority_class, size=len(majority_class), replace=True)

# Combine the majority class samples with the randomly duplicated minority class samples
oversampled_indices = np.concatenate([majority_class, random_indices])
X_oversampled = X[oversampled_indices]
y_oversampled = y[oversampled_indices]

# Check the class distribution
print(np.unique(y_oversampled, return_counts=True))
```

```
(array([0, 1]), array([65, 65], dtype=int64))
```

(65 samples with label 0, 65 samples with label 1)

Training and testing for all strategies:

```
X_train, X_test, Y_train, Y_test = train_test_split( X, y, test_size=0.30,
random_state=30)
lm = LinearRegression()
lm.fit(X_train, Y_train)
# model evaluation for training set
y_train_predict = lm.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

Y_pred = lm.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_pred)))
r2 = r2_score(Y_test, Y_pred)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))

# model evaluation for testing set
print(Y_pred.shape)
Y_pred

pd.DataFrame(Y_pred,Y_test)
```

For Unbalanced Data:

```
The model performance for training set
--------------------------------------
RMSE is 0.32118330558650104
R2 score is 0.04445627649852801


The model performance for testing set
--------------------------------------
RMSE is 0.3045933475029041
R2 score is -0.09390943379733496
```

For Over Sampled Data:

```
The model performance for training set
--------------------------------------
RMSE is 0.46795565829606034
R2 score is 0.1239836440092964


The model performance for testing set
--------------------------------------
RMSE is 0.4793744043026675
R2 score is 0.0803098679806089
```

For Under Sampled Data:

```
The model performance for training set
--------------------------------------
RMSE is 0.40078359281798837
R2 score is 0.34410442288894494


The model performance for testing set
--------------------------------------
RMSE is 0.5859752886795905
R2 score is -0.5451516752440824
```

Clearly, Over Sampling helps improve on the results but under sampling is not very useful in this case