Tebibu Kebede

Midterm 3

1

A.

i. reading

ii.

Classification report without PCA:

	precision	recall	f1-score	support
1	0.75	0.67	0.71	9
2	1.00	1.00	1.00	19
3	0.77	0.83	0.80	12
accuracy			0.88	40
macro avg	0.84	0.83	0.84	40
weighted avg	0.87	0.88	0.87	40

Classification report with PCA:

	precision	recall	f1-score	support
1	0.75	0.67	0.71	9
2	1.00	1.00	1.00	19
3	0.77	0.83	0.80	12
accuracy			0.88	40
macro avg	0.84	0.83	0.84	40
weighted avg	0.87	0.88	0.87	40

iii.

PCA Components:

```
[[ 0.44304849  0.44149753  0.27564393  0.42694661  0.4287223  -0.13163545
  0.38651556]
 [-0.03212855 -0.07897768  0.46886513 -0.17890281  0.09538415 -0.7678795
  -0.37703115]]
```

Explained variance ratio:

```
[0.70923426 0.18044611]
```

iv.

PCA-transformed training data:

```
[[-1.8556861   0.67631507]
 [-0.66543848  1.21814778]
 [ 1.17560536  1.71857122]
 [-1.48431251 -0.0654026 ]
 [-1.80509469 -0.01576124]
 [-1.02496994  1.71415541]
 [-0.50208967  0.21516212]
 [-2.76834158 -1.13178281]
 [ 0.73502831 -0.84107347]
 [ 2.88317511  0.58580103]
 [ 0.59771348  1.32750689]
 [-0.27075282  1.60816451]
 [ 2.65448565  0.77756715]
 [ 3.18447245 -0.06398599]
 [ 3.41740201 -0.46842496]
 [-2.94172179 -2.17115134]
 [ 3.09003992  0.17269764]
 [-2.83395747 -0.29223337]
 [-1.39862135 -1.58176673]
 [-3.25053772 -0.76544396]
 [-2.530508    0.39044077]
 [-0.43218834 -0.44308733]
 [-0.85421715  0.87155084]
 [ 1.90955898  1.01558493]
 [ 3.76381252 -1.68448781]
 [ 0.46627831  1.09024997]
 [ 4.20809978 -1.56623543]
 [-1.21738476 -1.21125775]
 [-1.07526971 -1.93250488]
 [-0.27835425  0.62757421]
 [ 2.89450179  0.70805136]
 [-1.78343619 -0.38629697]
 [ 0.7812092  -0.50048773]
 [ 4.03744256 -0.56958055]
 [ 4.4273685  -1.34030297]
 [-1.62914459  0.82996917]
 [ 0.78239163 -0.1426398 ]
 [ 2.53274269  0.97439829]
 [-3.17298582 -1.71944303]
 [-0.39520683  1.83716045]
 [ 2.7755252  -0.82558728]
 [-0.03035064  1.0037091 ]
```

[3.45014379 -0.19121884]
[-1.9366198   0.11972907]
[-2.41888881 -1.00696034]
[0.19871992 0.60062721]
[-1.47481324 1.9750882]
[-1.89794278 1.67931062]
[-2.84595311 -0.81693759]
[3.33865854 -0.43883384]
[0.46641696 0.96225464]
[-0.87898878 0.8016866]
[-1.95229962 0.08869743]
[-2.93828919 -1.16754498]
[-0.09256693 0.40655144]
[-0.39271983 1.04752886]
[-1.54755575 0.30180603]
[1.17677694 2.07341737]
[-0.53311375 1.17845265]
[3.21745943 -0.44676374]
[-2.15285324 -0.99060652]
[-1.93289484 0.82597557]
[-3.15217937 -1.86848925]
[-0.49826245 1.60887205]
[-1.54624005 1.89885988]
[0.838404   -0.87455349]
[-2.44620957 -2.49353943]
[0.38331629 0.84578346]
[-3.12617831 -0.37443689]
[-2.49140645 -1.53062278]
[1.70816001 -0.52685593]
[0.69172199 1.25305275]
[-0.48571315 0.81441475]
[0.59004203 1.7138169]
[-3.13397627 -1.01694673]
[-1.02776604 0.147916]
[-1.11188404 1.98385952]
[2.911986   0.68411081]
[-1.42041227 -0.26972544]
[-1.94834207 -0.54463885]
[-1.49794332 0.57335369]
[0.07077786 1.11211698]
[-2.49618487 0.52554946]
[-1.94260773 -1.21758367]
[2.98214157 0.11380948]
[2.80190669 -2.34632215]

[3.7795297   0.27185964]
[4.49142153 -1.58358089]
[-0.66012298 1.35799136]
[-2.88457735 -1.16528479]
[0.19267407 0.76030143]
[2.70788113 -0.71261275]
[-2.62417612 -1.61242071]
[-0.70560611 1.74091146]
[-1.86383492 -0.71544527]
[-0.52359571 1.90256585]
[2.85218878 0.68732092]
[0.28866576 0.72982917]
[1.51175963 -0.25548471]
[-2.28148909 0.13290622]
[-2.32826933 -0.85646402]
[-3.18541355 -1.112371]
[0.42263517 -0.3681301]
[3.5741644 -0.42123956]
[1.12088233 0.32022394]
[1.83153016 0.37991601]
[3.5074214   0.09815613]
[1.81783155 -0.8046508]
[3.4600173 -1.52404036]
[-1.4024645 -1.51597429]
[-3.23345669 0.02143547]
[2.78494454 -1.00081244]
[1.50751399 1.0422699]
[-0.94493701 0.97333975]
[-1.90560873 1.56534073]
[-1.69845772 0.82769332]
[0.85846321 -0.35398332]
[-0.16148916 2.72310057]
[1.72863029 -0.94139208]
[-2.4153819 -0.62989929]
[-1.31409249 -1.24834946]
[-2.86878111 -0.2326707]
[-2.62257795 0.10134516]
[-0.13694364 1.46836603]
[-0.25442028 1.10093628]
[3.49285845 0.86721921]
[-0.31928898 -0.63606322]
[-2.67539866 -0.20577554]
[-2.55568604 1.89596548]
[-0.27727536 0.16647189]

```
 [ 3.13326029 -0.16610596]
 [ 0.93809702  2.3644041 ]
 [-2.32308549  1.40977807]
 [-1.86475833 -0.92565668]
 [-1.96843294 -0.67010465]
 [ 0.94766092 -0.83434163]
 [-0.65650151 -0.65479478]
 [-1.67547498 -0.38350115]
 [-1.6566626  -0.24790383]
 [ 0.07649469  1.99172387]
 [ 0.03744122  0.70106002]
 [-2.92359547 -1.75480808]
 [ 2.24215354 -0.42903748]
 [-2.44686529 -1.67723653]
 [ 4.19333899 -1.0614086 ]
 [ 3.28802656 -0.81174988]
 [ 2.39462541 -0.5329564 ]
 [ 3.21870751  0.12479502]
 [ 3.22042204 -1.77076056]
 [ 2.73806127 -1.26259321]
 [-1.92003285 -0.62565378]
 [-0.41403799  0.73594225]
 [-1.05511626  0.16087553]
 [ 1.30260933 -1.25105649]
 [ 3.17253307 -0.73237325]
 [-0.46750087  0.93835159]
 [ 3.52627004 -0.63900634]
 [-2.94012897 -1.64638288]
 [ 2.17271418 -0.44621977]]
```

B.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Load dataset
file_path = 'C:/Users/tebib/OneDrive/Desktop/J2/Topics in intelligent systems/Midterm
3/seeds.csv'
data = pd.read_csv(file_path, delimiter=',')


X = data.drop('Type', axis=1)
```

```
y = data['Type']


X_standardized = StandardScaler().fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X_standardized, y, test_size=0.2,
random_state=42)

# Train SVM without PCA
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)
print("SVM Classification report without PCA:")
print(classification_report(y_test, y_pred))

# Apply PCA to reduce dimensions to 3 features
pca = PCA(n_components=3)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
svm_classifier_pca = SVC()
svm_classifier_pca.fit(X_train_pca, y_train)
y_pred_pca = svm_classifier_pca.predict(X_test_pca)
print("SVM Classification report with PCA:")
print(classification_report(y_test, y_pred_pca))

# Comparing accuracy
accuracy_without_pca = accuracy_score(y_test, y_pred)
accuracy_with_pca = accuracy_score(y_test, y_pred_pca)

print(f'Accuracy without PCA: {accuracy_without_pca}')
print(f'Accuracy with PCA: {accuracy_with_pca}')


SVM Classification report without PCA:
              precision    recall  f1-score   support

           1       0.75      0.67      0.71         9
           2       1.00      1.00      1.00        19
           3       0.77      0.83      0.80        12

    accuracy                           0.88        40
   macro avg       0.84      0.83      0.84        40
weighted avg       0.87      0.88      0.87        40
```

SVM Classification report with PCA:

	precision	recall	f1-score	support
1	0.75	0.67	0.71	9
2	1.00	1.00	1.00	19
3	0.77	0.83	0.80	12
accuracy			0.88	40
macro avg	0.84	0.83	0.84	40
weighted avg	0.87	0.88	0.87	40

Accuracy without PCA: 0.875
Accuracy with PCA: 0.875

C.

PCA helps in reducing the number of features while retaining the most significant information in the dataset. This can lead to a simpler model that is faster to train and may generalize better to new data.By limiting the number of features, PCA can help reduce the risk of overfitting, especially if there's multicollinearity in the features, or if the dataset has many features but not a lot of records. With a reduced number of dimensions (for example, down to 2 or 3), it becomes possible to visualize high-dimensional data in a 2D or 3D space, which can offer insights into the structure of the data, the relationships between features, and how the classes are separated. Fewer dimensions can lead to less computational overhead when training models, as there's less data to process, and algorithms can run faster.

2. import numpy as np

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from scipy.stats import pearsonr

# Given data
x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]).reshape((-1, 1))
y = np.array([99, 90, 87, 88, 111, 91, 103, 87, 94, 78, 77, 85, 86])

# (a) Solve and Print the Simple Linear Regression Equation
model = LinearRegression().fit(x, y)
intercept = model.intercept_
slope = model.coef_[0]
```

```
regression_eq = f"y = {slope:.2f}x + {intercept:.2f}"

# (b) Solve and Print the Pearson Correlation Coefficient (r)
pearson_r, _ = pearsonr(x.flatten(), y)

# (c) Solve and Print the RMSE or std_error
y_pred = model.predict(x)
rmse = np.sqrt(mean_squared_error(y, y_pred))

# (d) Predict the Speed of a Car with age 10 years old
predicted_speed = model.predict(np.array([[10]]))[0]

regression_eq, pearson_r, rmse, predicted_speed
```

(a) The Simple Linear Regression Equation is: $y = -1.54x + 102.18$

(b) The Pearson Correlation Coefficient (r) is: $r = -0.6766$

(c) The Root Mean Squared Error (RMSE) or std_error is: RSME=6.72

(d) The predicted speed of a car with an age of 10 years old is: Speed= 86.79

3. The calculated r2-coefficient is approximately 0.874, and the Root Mean Squared Error (RMSE) is approximately 3.23. This indicates that the model fits the data well and has a relatively low error in the predictions.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
import numpy as np

# Given dataset
x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]).reshape(-1, 1)
y = np.array([99, 90, 87, 88, 111, 91, 103, 87, 94, 78, 77, 85, 86])

# Transforming data to include polynomial features of degree 3
polynomial_features = PolynomialFeatures(degree=3)
x_poly = polynomial_features.fit_transform(x)

# Performing Linear Regression on the transformed data
model = LinearRegression()
model.fit(x_poly, y)
y_poly_pred = model.predict(x_poly)
```

```
# Calculating r2 coefficient and RMSE
r2 = r2_score(y, y_poly_pred)
rmse = np.sqrt(mean_squared_error(y, y_poly_pred))

r2, rmse
```

4. Average RMSE: 0.9523249454213989