Fundamentals of Data Science - Project report

Mateusz Cichostępski

January 2023

Contents

1 Statistical summary of wine dataset

The wine dataset contains the results of a chemical analysis of wines grown in the same region in Italy, but derived from three different cultivars. Total number of instances is 178. The analysis determined the quantities of 13 constituents found in each of three types of wines. The attributes of the dataset are:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

All attributes are continuous. The 1st attribute of rows is class identifier. All of the attributes are numeric, mostly decimal.

These are the first 5 rows of the dataset:

1	14.23	1.71	2.43	15.6	127	2.8	3.06	.28	2.29	5.64	1.04	3.92	1065
1	13.2	1.78	2.14	11.2	100	2.65	2.76	.26	1.28	4.38	1.05	3.4	1050
1	13.16	2.36	2.67	18.6	101	2.8	3.24	.3	2.81	5.68	1.03	3.17	1185
1	14.37	1.95	2.5	16.8	113	3.85	3.49	.24	2.18	7.8	.86	3.45	1480
1	13.24	2.59	2.87	21	118	2.8	2.69	.39	1.82	4.32	1.04	2.93	735

The mean of the features grouped by classification are following:

Class	1	2	3
Alcohol	13.744746	12.278732	13.153750
Malic acid	2.010678	1.932676	3.333750
Ash	2.455593	2.244789	2.437083
Alcalinity of ash	17.037288	20.238028	21.416667
Magnesium	106.338983	94.549296	99.312500
Total phenols	2.840169	2.258873	1.678750
Flavanoids	2.982373	2.080845	0.781458
Nonflavanoid phenols	0.290000	0.363662	0.447500
Proanthocyanins	1.899322	1.630282	1.153542
Color intensity	5.528305	3.086620	7.396250
Hue	1.062034	1.056282	0.682708
OD280/OD315 of diluted wines	3.157797	2.785352	1.683542
Proline	1115.711864	519.507042	629.895833

Most of the averages between classes are similar. The diffrence can be noticed in:

- *Malic acid* - 3rd class outliers from other

- *Flavanoids* - 3rd class outliers from other
- *Color intensity* - 2nd class outliers from other
- *OD280/OD315 of diluted wines* - 3rd class outliers from other
- *Proline* - 1st class outliers from other

2 Reducing data dimensionality

Reduction of data dimensionality was carried out using PCA.

```
pca = PCA(n_components=2)
pca.fit(df_without_class)
df_pca = pd.DataFrame(pca.transform(df_without_class))
```

The number of components to keep was chosen to 2, because of varianced ratio which was:

$$\begin{bmatrix} 0.99809123 & 0.00173592 \end{bmatrix} \tag{1}$$

3 Visualization of reduced dataset

To reduced DataFrame was inserted column with classification labels. Then dataset was grouped by in order to differentiate classes on plot.

```
df_pca.insert(2, 2, list(df_classes))

df_pca.insert(2, 2, list(df_classes))
groups = df_pca.groupby(2)
```

Finally, the generated plot was saved to file (figure 1).

```
for name, group in groups:
  plt.plot(group[0], group[1], marker="o", linestyle="", label=name)
plt.legend()
plt.savefig("reducedData.png")
```

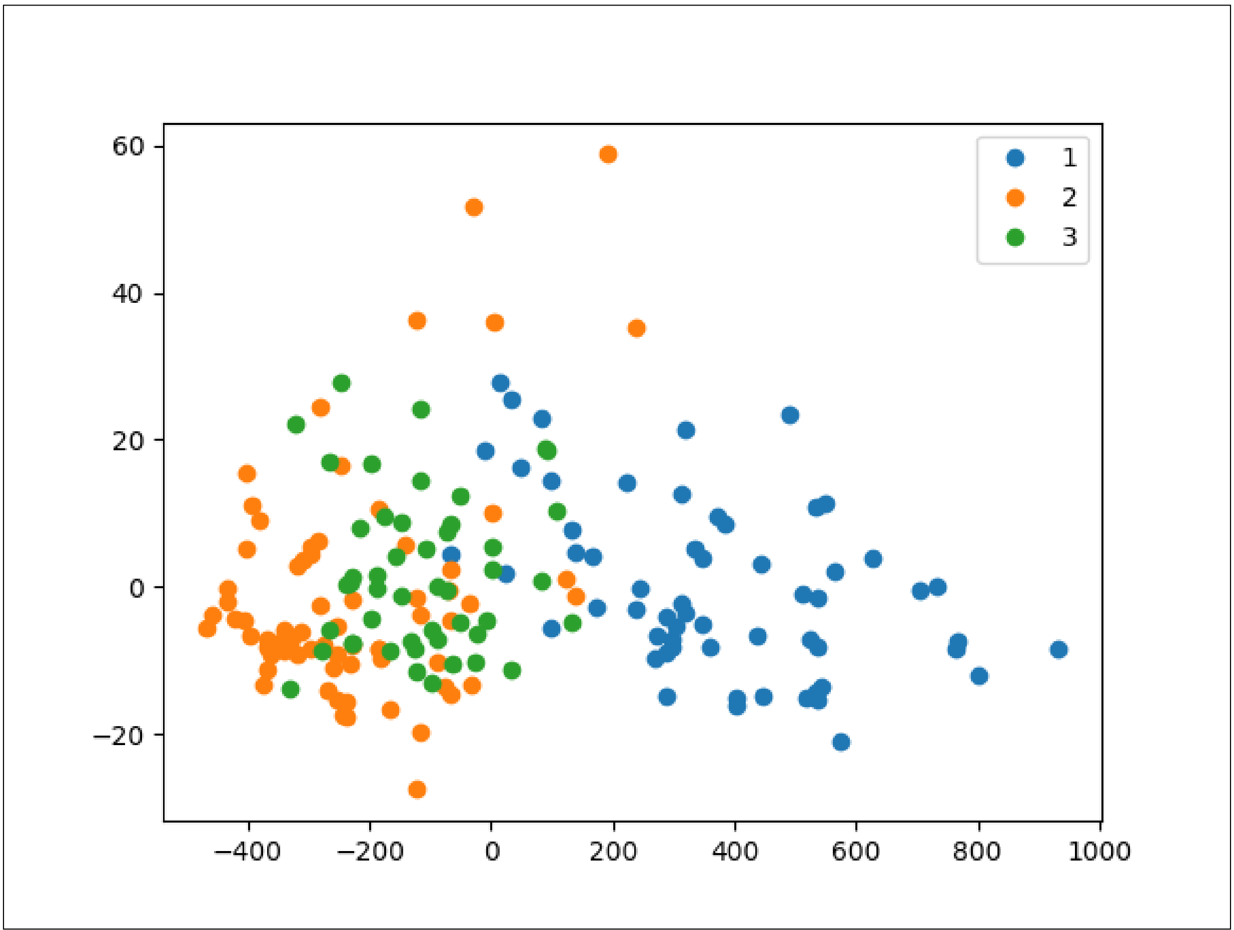

Figure 1: Visualization of reduced dataset

The visible dissimilarity is between class 1 and others. It is hard to see significant difference between class 2 and 3.

4 Clustering the dataset

Clustering was performed using KMeans Algorithm on dataset before and after dimension reduction.

```
kmeans_clustering = KMeans(
  n_clusters=3, n_init='auto').fit_predict(df_without_class)

kmeans_clustering = KMeans(
  n_clusters=3, n_init='auto').fit_predict(df_pca)
```

Then using *rand_score* function the accuracy of clustering was evaluated.

```
print(f'Kmeans accurate: {rand_score(kmeans_clustering, df_classes)}')
```

Dataset	Accuracy
Before PCA	0.71866
After PCA	0.69187

5 Splitting the dataset

Splitting the dataset into training and testing subsets was carried out using *train_test_split* function. The ratio was 33% of testing set and 67% training set.

```
X_train, X_test, y_train, y_test = train_test_split(
  df_without_class, df_classes, test_size=0.33, random_state=121)
```

6 Performing classification

Classification was performed using k-nearest neighbors algorithm with number of neighbors k from 1 to 10 and 3 distance computations (*cosine*, *euclidean*, *manhattan*). The accuracy of performed classifications was evaluated by score function and show in the table below.

```
classificationAccuracy = {}

for k in range(1, 11):
  for met in ['cosine', 'euclidean', 'manhattan']:
    knn = KNeighborsClassifier(n_neighbors=k, metric=met)
    knn.fit(X_train, y_train)

    if met in classificationAccuracy.keys():
      classificationAccuracy[met].append(knn.score(X_test, y_test))
    else:
      classificationAccuracy[met] = [knn.score(X_test, y_test)]
```

n_neighbors	**cosine**	**euclidean**	**manhattan**
1	0.813559	0.677966	0.745763
2	0.762712	0.627119	0.661017
3	0.711864	0.728814	0.728814
4	0.779661	0.644068	0.694915
5	0.728814	0.627119	0.677966
6	0.745763	0.593220	0.644068
7	0.711864	0.627119	0.677966
8	0.677966	0.627119	0.644068
9	0.661017	0.661017	0.694915
10	0.644068	0.644068	0.677966
Avg	0.723729	0.645763	0.684746

Cosine metric has the best average accuracy. After it there are *manhattan* and *euclidean* distance computations. For the final choice I would choose *cosine* metric with $k = 4$ nearest neighbors.

7 Student vs ChatGPT

7.1 Load dataset

```
#->Using python load the wine dataset to dataFrame.

data = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.
   data", header=None)
```

First task to *chatGPT* was to load dataset, which was done correctly. AI did not set header labels.

7.2 Statistical summary

```
#->Do statistical summary of the data and discuss it.

print(data.describe())
print(data.info())
```

ChatGPT used built-in function to describe statistical summary. It was better solution than mine. But it was harder to analyze summary due to the lack of attribute labels. Additionally *chatGPT* included also column with class identifier, which was not unnecessary.
AI used *info()* function, which basicaly prints type of variable and types of included columns.

7.3 Dimensionality reduction

```
#->Reduce data dimensionality

# Create a PCA object with the desired number of components
pca = PCA(n_components=3)

# Fit the PCA model to the data
pca.fit(data)

# Transform the data to the new lower-dimensional space
data_pca = pca.transform(data)
```

ChatGPT also used PCA algorithm. But it made a huge mistake, it has included column with class identifier to perform PCA. Another difference was that it set number of components to keep as 3.

To be fair I have asked chat to perform PCA once again, but this time without first column.

```
#->Create a copy of the wine dataset and remove the first
#   column with the class information, then reduce data dimensionality using
    PCA.

# create a copy of the dataset without the first column
data_copy = data.iloc[:, 1:]

# initialize PCA
pca = PCA(n_components=2)

# fit PCA on the data
pca.fit(data_copy)

# transform the data to the first two principal components
data_pca = pca.transform(data_copy)
```

This time *chatGPT* set number of components to keep as 2. The task was carried out with the same way as me.

7.4 Visualize data

```
#->Visualize the reduced dataset.

# create scatter plot of the reduced data
plt.scatter(data_pca[:, 0], data_pca[:, 1])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()

class_col = data.iloc[:, 0]
colors = ['r', 'b', 'g']
for i in range(len(class_col)):
  plt.scatter(data_pca[i, 0], data_pca[i, 1], c=colors[int(class_col[i]) -
    1])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```

ChatGPT did it with similar way as me. Plots look identically. These are the generated plots:

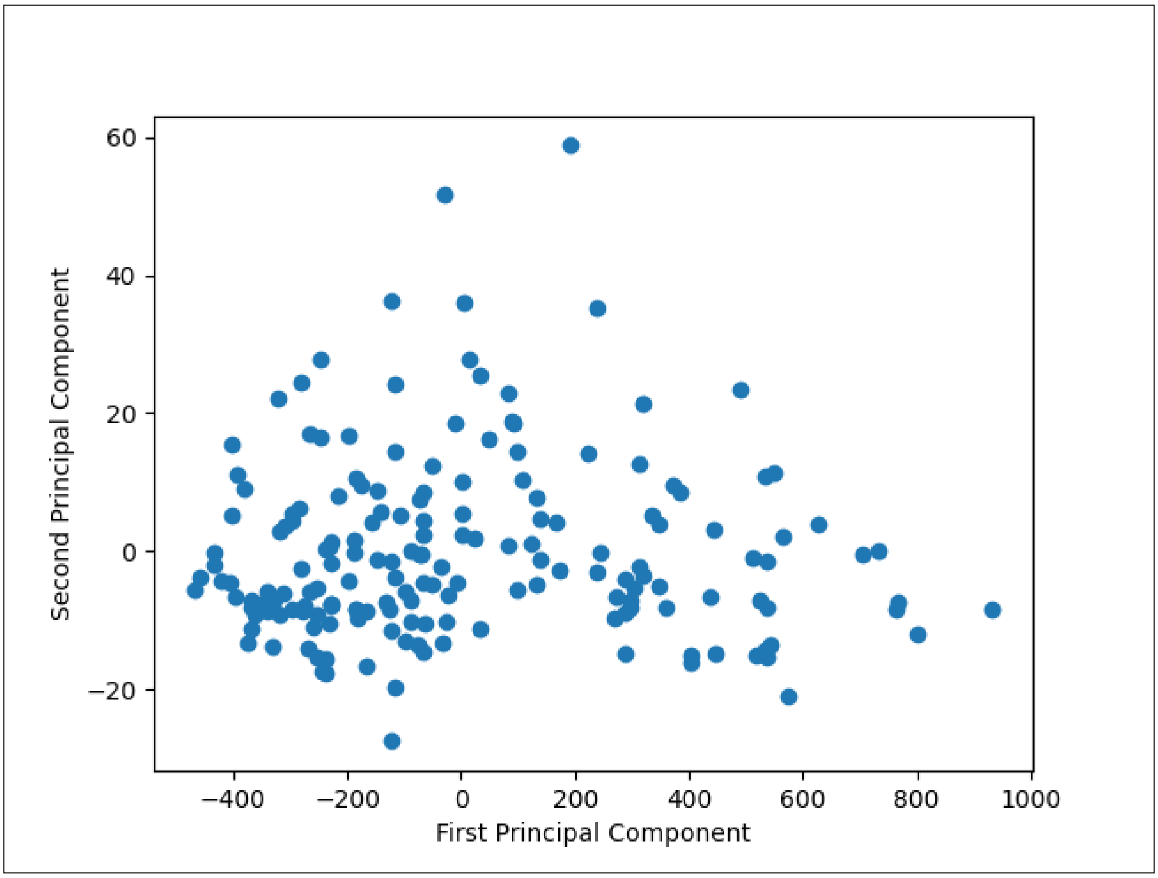

Figure 2: Visualization of reduced dataset by *chatGPT*

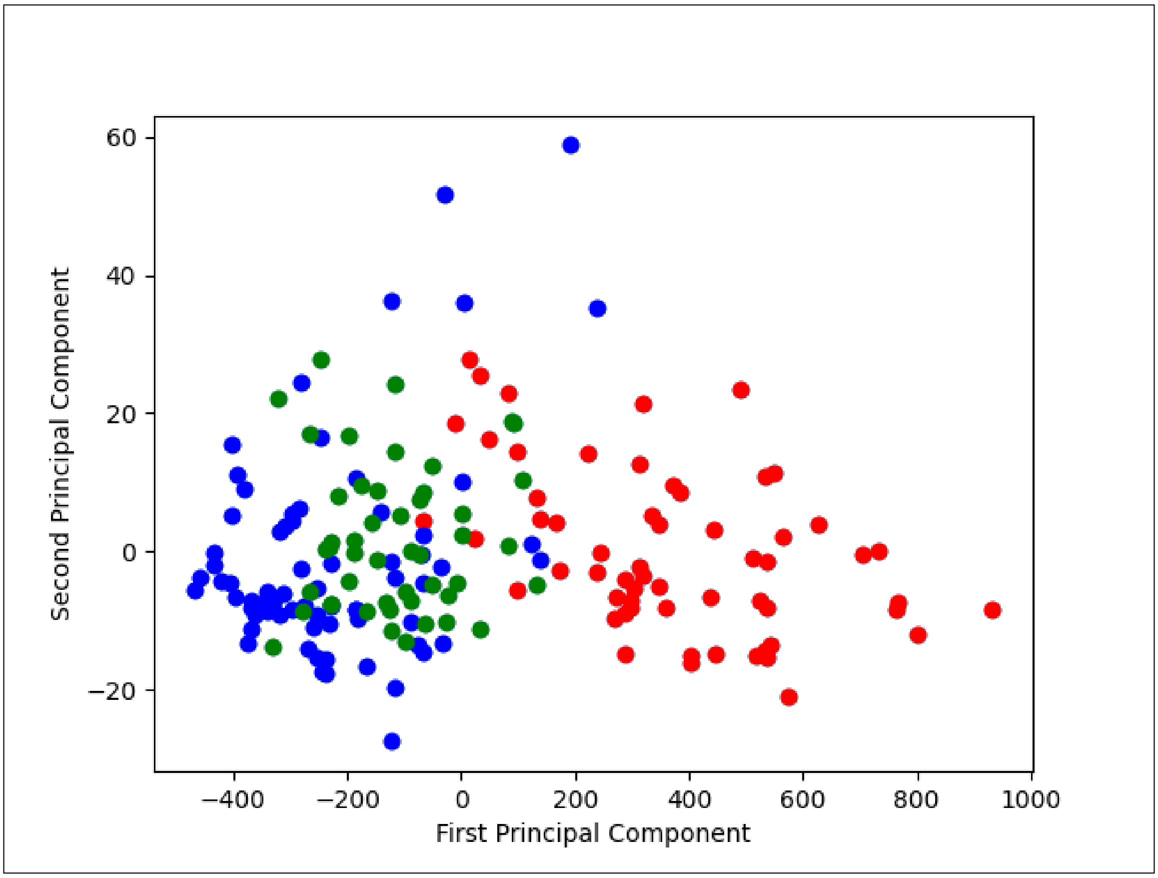

Figure 3: Visualization of reduced dataset with coloring labels by *chatGPT*

7.5 Cluster the dataset

```
#->Cluster the dataset (and evaluate clustering results with classification
    labels).

# initialize KMeans
kmeans = KMeans(n_clusters=3)

# fit KMeans on the reduced data
kmeans.fit(data_pca)

# predict the cluster labels
labels = kmeans.predict(data_pca)


# evaluate the clustering results using the adjusted Rand index
ari = adjusted_rand_score(class_col, labels)
print("Adjusted Rand index: ", ari)
```

Again *chatGPT* did it with similar way. There are some differences. It have not set the value of *n_init* attribute, it causes warnings in command line. 2nd difference was evaluating the clustering results. It have used *adjusted_rand_score()* function, which returns a index in range $[-0.5, 1]$. It is less intuitive than *rand_score()* $[0, 1]$.

7.6 Split the dataset into training and testing

```
#->Split the dataset into training and testing.

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
  data.iloc[:, 1:], data.iloc[:, 0], test_size=0.2, random_state=42)
```

Again the same solution of this task, except proportion between subsets and *random_state*.

7.7 Perform classification

7.7.1 First attempt

```
#->Perform classification and evaluate its result.

# initialize logistic regression
clf = LogisticRegression()

# fit logistic regression on the training data
clf.fit(X_train, y_train)

# make predictions on the testing data
y_pred = clf.predict(X_test)

# calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
```

At the first attempt *chatGPT* has used Logistic Regression classifier, which we have not discussed during course. During classification appeared warning about iterations limit. Overall the accuracy score was around 97%.

7.7.2 Second attempt

I have asked AI to do this task using KNN algorithm.

```
#->Perform classification using KNN and evaluate its result.

# initialize KNN
knn = KNeighborsClassifier(n_neighbors=5)

# fit KNN on the training data
knn.fit(X_train, y_train)

# make predictions on the testing data
y_pred = knn.predict(X_test)

# calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
```

ChatGPT has chosen $k = 5$ numbers of n_neighbors and default distance computation. Accuracy of this classification was around 72.2%.

8 Summary

Implementation of this project using *chatGPT* is practically identical to implementations during classes. Unfortunately, or fortunately (for us), AI did some minor or major mistakes that only user familiar with task can fix.