Wine Analysis

November 9, 2025

1 Coding 4:

Group Members:

Zubair Lalani (zubairl2)

Adithya Swaminathan (adithya9)

1.1 Setup

Import necessary packages to perform the analysis

[1]:
```python
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis,␣
 ↪LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

1.2 Problem 1: Wine Classification

1.2.1 Importing Data

[2]:
```python
# Feature names taken from the "wine.names" file that came with the data

col_names = [
    "class_num",
    "alcohol",
    "malic_acid",
    "ash",
    "alcalinity_of_ash",
    "magnesium",
    "total_phenols",
    "flavanoids",
    "nonflavanoid_phenols",
    "proanthocyanins",
    "color_intensity",
    "hue",
    "od280_od315_of_diluted_wines",
    "proline",
]

df = pd.read_csv("data/wine/wine.data", header=None, names=col_names)
```

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

[3]:
```
   class_num  alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  \
0          1    14.23        1.71  2.43               15.6        127
1          1    13.20        1.78  2.14               11.2        100
2          1    13.16        2.36  2.67               18.6        101
3          1    14.37        1.95  2.50               16.8        113
4          1    13.24        2.59  2.87               21.0        118

   total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
0           2.80        3.06                  0.28             2.29
1           2.65        2.76                  0.26             1.28
2           2.80        3.24                  0.30             2.81
3           3.85        3.49                  0.24             2.18
4           2.80        2.69                  0.39             1.82

   color_intensity   hue  od280_od315_of_diluted_wines  proline
0             5.64  1.04                          3.92     1065
1             4.38  1.05                          3.40     1050
2             5.68  1.03                          3.17     1185
3             7.80  0.86                          3.45     1480
4             4.32  1.04                          2.93      735
```

1.2.2 Preprocessing

There are no missing values or duplicate rows. Seems no preprocessing is needed

[4]:
```
df.isna().sum().sort_values(ascending=False)
```

[4]:
```
class_num                       0
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
```

```
proanthocyanins                 0
color_intensity                 0
hue                             0
od280_od315_of_diluted_wines    0
proline                         0
dtype: int64
```

[5]:
```python
df.duplicated().sum()
```

[5]:
```
0
```

1.2.3 Train/Test Split

[6]:
```python
y = df.iloc[:, 0]
X= df.iloc[:, 1:]
```

Check class balance and shapes of data

[7]:
```python
print(y.value_counts())
print(X.shape, y.shape)
```

```
class_num
2    71
1    59
3    48
Name: count, dtype: int64
(178, 13) (178,)
```

Perform stratified 70-30 train/test split wtih seed 598

[8]:
```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,
    random_state=598,
    stratify=y
)
```

[9]:
```python
print(y_train.value_counts())
print(y_test.value_counts())
```

```
class_num
2    50
1    41
3    33
Name: count, dtype: int64
class_num
2    21
1    18
```

```
3    15
Name: count, dtype: int64
```

Split with no stratification as well (as a sanity check since we get 100% accuracy in later sections)

[10]:
```
X_train_no_strat, X_test_no_strat, y_train_no_strat, y_test_no_strat =␣
↪train_test_split(
    X, y,
    test_size=0.30,
    random_state=598,
)
```

[11]:
```
print(y_train_no_strat.value_counts())
print(y_test_no_strat.value_counts())
```

```
class_num
2    45
3    43
1    36
Name: count, dtype: int64
class_num
2    26
1    23
3     5
Name: count, dtype: int64
```

1.2.4 Part A

Using the training data fit three models: LDA, QDA and Multinomial (Logistic) regression. Note that here the response has three levels, so a binomial logistic is not applicable.

QDA:

[12]:
```
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)



qda_no_strat = QuadraticDiscriminantAnalysis()
qda_no_strat.fit(X_train_no_strat, y_train_no_strat)
```

[12]:
```
QuadraticDiscriminantAnalysis()
```

LDA:

[13]:
```
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

lda_no_strat = LinearDiscriminantAnalysis()
lda_no_strat.fit(X_train_no_strat, y_train_no_strat)
```

```
[13]: LinearDiscriminantAnalysis()
```

Multinomial Logisitc Regression

```
[14]: # should do multinomial automatically by default
      # Scaling and increased max_iter to ensure convergence
      mlr = make_pipeline(
          StandardScaler(with_mean=True),
          LogisticRegression(max_iter=2000, solver="lbfgs", n_jobs=-1)  # bump␣
        ↪max_iter
      )

      mlr.fit(X_train, y_train)


      mlr_no_strat = make_pipeline(
          StandardScaler(with_mean=True),
          LogisticRegression(max_iter=2000, solver="lbfgs", n_jobs=-1)  # bump␣
        ↪max_iter
      )

      mlr_no_strat.fit(X_train_no_strat, y_train_no_strat)
```

```
[14]: Pipeline(steps=[('standardscaler', StandardScaler()),
                      ('logisticregression',
                       LogisticRegression(max_iter=2000, n_jobs=-1))])
```

1.2.5 Part B

Report the accuracy for all methods on both training and testing data sets, and prepare confusion matrices.

```
[15]: # Accuracy on training data

      qda_predictions_train = qda.predict(X_train)
      lda_predictions_train = lda.predict(X_train)
      mlr_predictions_train = mlr.predict(X_train)

      qda_accuracy_train = accuracy_score(y_train, qda_predictions_train)
      lda_accuracy_train = accuracy_score(y_train, lda_predictions_train)
      mlr_accuracy_train = accuracy_score(y_train, mlr_predictions_train)

      # Confusion matrix for training data
      cm_qda_train = confusion_matrix(y_train, qda_predictions_train)
      cm_lda_train = confusion_matrix(y_train, lda_predictions_train)
      cm_mlr_train = confusion_matrix(y_train, mlr_predictions_train)
```

[16]:
```
print("(train) QDA Accuracy: ", qda_accuracy_train)
print("(train) LDA Accuracy: ", lda_accuracy_train)
print("(train) MLR Accuracy: ", mlr_accuracy_train)
```

```
(train) QDA Accuracy:  1.0
(train) LDA Accuracy:  1.0
(train) MLR Accuracy:  1.0
```

[17]:
```
print("(train) QDA Confusion Matrix\n", cm_qda_train)
print("(train) LDA Confusion Matrix\n", cm_lda_train)
print("(train) MLR Confusion Matrix\n", cm_mlr_train)
```

```
(train) QDA Confusion Matrix
 [[41  0  0]
 [ 0 50  0]
 [ 0  0 33]]
(train) LDA Confusion Matrix
 [[41  0  0]
 [ 0 50  0]
 [ 0  0 33]]
(train) MLR Confusion Matrix
 [[41  0  0]
 [ 0 50  0]
 [ 0  0 33]]
```

[18]:
```
qda_predictions_test = qda.predict(X_test)
lda_predictions_test = lda.predict(X_test)
mlr_predictions_test = mlr.predict(X_test)

qda_accuracy_test = accuracy_score(y_test, qda_predictions_test)
lda_accuracy_test = accuracy_score(y_test, lda_predictions_test)
mlr_accuracy_test = accuracy_score(y_test, mlr_predictions_test)

# Confusion matrix for testing data
cm_qda_test = confusion_matrix(y_test, qda_predictions_test)
cm_lda_test = confusion_matrix(y_test, lda_predictions_test)
cm_mlr_test = confusion_matrix(y_test, mlr_predictions_test)
```

[19]:
```
print("(test) QDA Accuracy: ", qda_accuracy_test)
print("(test) LDA Accuracy: ", lda_accuracy_test)
print("(test) MLR Accuracy: ", mlr_accuracy_test)
```

```
(test) QDA Accuracy:  1.0
(test) LDA Accuracy:  1.0
(test) MLR Accuracy:  1.0
```

[20]:
```
print("(test) QDA Confusion Matrix\n", cm_qda_test)
print("(test) LDA Confusion Matrix\n", cm_lda_test)
print("(test) MLR Confusion Matrix\n", cm_mlr_test)
```

```
(test) QDA Confusion Matrix
 [[18  0  0]
 [ 0 21  0]
 [ 0  0 15]]
(test) LDA Confusion Matrix
 [[18  0  0]
 [ 0 21  0]
 [ 0  0 15]]
(test) MLR Confusion Matrix
 [[18  0  0]
 [ 0 21  0]
 [ 0  0 15]]
```

As a sanity check, also tried without stratification to see if we still get 100% accuracy

[21]:
```
qda_predictions_no_strat_train = qda_no_strat.predict(X_train_no_strat)
lda_predictions_no_strat_train = lda_no_strat.predict(X_train_no_strat)
mlr_predictions_no_strat_train = mlr_no_strat.predict(X_train_no_strat)

qda_accuracy_no_strat_train = accuracy_score(y_train_no_strat,␣
  ↪qda_predictions_no_strat_train)
lda_accuracy_no_strat_train = accuracy_score(y_train_no_strat,␣
  ↪lda_predictions_no_strat_train)
mlr_accuracy_no_strat_train = accuracy_score(y_train_no_strat,␣
  ↪mlr_predictions_no_strat_train)

print("(train) QDA (no strat) Accuracy: ", qda_accuracy_no_strat_train)
print("(train) LDA (no strat) Accuracy: ", lda_accuracy_no_strat_train)
print("(train) MLR (no strat) Accuracy: ", mlr_accuracy_no_strat_train)


qda_predictions_no_strat_test = qda_no_strat.predict(X_test_no_strat)
lda_predictions_no_strat_test = lda_no_strat.predict(X_test_no_strat)
mlr_predictions_no_strat_test = mlr_no_strat.predict(X_test_no_strat)

qda_accuracy_no_strat_test = accuracy_score(y_test_no_strat,␣
  ↪qda_predictions_no_strat_test)
lda_accuracy_no_strat_test = accuracy_score(y_test_no_strat,␣
  ↪lda_predictions_no_strat_test)
mlr_accuracy_no_strat_test = accuracy_score(y_test_no_strat,␣
  ↪mlr_predictions_no_strat_test)

print("(test) QDA (no strat) Accuracy: ", qda_accuracy_no_strat_test)
print("(test) LDA (no strat) Accuracy: ", lda_accuracy_no_strat_test)
print("(test) MLR (no strat) Accuracy: ", mlr_accuracy_no_strat_test)

cm_qda_test = confusion_matrix(y_test_no_strat, qda_predictions_no_strat_test)
```

```
cm_lda_test = confusion_matrix(y_test_no_strat, lda_predictions_no_strat_test)
cm_mlr_test = confusion_matrix(y_test_no_strat, mlr_predictions_no_strat_test)

print("(test + no strat) QDA Confusion Matrix\n", cm_qda_test)
print("(test + no strat) LDA Confusion Matrix\n", cm_lda_test)
print("(test + no strat) MLR Confusion Matrix\n", cm_mlr_test)
```

```
(train) QDA (no strat) Accuracy:  1.0
(train) LDA (no strat) Accuracy:  1.0
(train) MLR (no strat) Accuracy:  1.0
(test) QDA (no strat) Accuracy:  0.9814814814814815
(test) LDA (no strat) Accuracy:  0.9629629629629629
(test) MLR (no strat) Accuracy:  0.9629629629629629
(test + no strat) QDA Confusion Matrix
 [[23  0  0]
 [ 1 25  0]
 [ 0  0  5]]
(test + no strat) LDA Confusion Matrix
 [[23  0  0]
 [ 0 25  1]
 [ 0  1  4]]
(test + no strat) MLR Confusion Matrix
 [[23  0  0]
 [ 0 25  1]
 [ 0  1  4]]
```

1.2.6 Part C

Comment on the results. For example, discuss the following: Is there a model that seems to perform better overall? Which model misclassified more observations? Is there a class that performed better or worse (in terms of classification)?

All of the models performed quite perfectly because the data is already well separated. In fact, all 3 of the models were able to achieve 100% accuracy on both the training and testing data. As a result, the confusion matrices only have numbers on the diagonals with the values equal to the class distribution between the three classes.

In order to sanity to check this result, we also tried performing the same steps without stratifying the dataset. In this case, we got relatively more "normal" results where we got 100% accuracy on the training set, but ~96%-98% accuracy on the testing dataset. Looking at the confusion matrices, the LDA + Logistic regression had 2 samples where it got confused between classes 2 and 3. The QDA model got confused between classes 1 and 2 once. With stratification, this confusion does not occur.

1.3 Problem 2: Digit Classification

Goal: goal is to compare the performance of SVM, Decision Trees and Boosting methods, such as AdaBoost, Gradient Boosting, XGBoost for correctly classifying all 10 digits

1.3.1 Importing Data

1.3.2 Train/Test Split

Perform 70-30 train/test split wtih seed 598

Do not filter out certain labels from the data like we did in previous assignments

[22]:
```
DATAFOLDER = "data/pen+based+recognition+of+handwritten+digits (1)/"
TRAIN_FILENAME = "pendigits.tra"
TEST_FILENAME = "pendigits.tes"
train_path = DATAFOLDER + TRAIN_FILENAME
test_path = DATAFOLDER + TEST_FILENAME


# Reading in the data
train_data = np.loadtxt(train_path, delimiter=",")
test_data = np.loadtxt(test_path, delimiter=",")

X_train = train_data[:, 0:16]
y_train = train_data[:, 16]
X_test = test_data[:, 0:16]
y_test = test_data[:, 16]
```

1.3.3 Part A

Fit a SVM classifier with a Gaussian kernel (also known as radial basis function) on the training data set.

[23]:
```
svm_classifier = SVC(kernel='rbf', random_state=598)
svm_classifier.fit(X_train, y_train)
y_pred_svm_train, y_pred_svm_test = svm_classifier.predict(X_train),␣
 ↪svm_classifier.predict(X_test)
```

1.3.4 Part B

Fit a Decision Tree classifier on the training data set.

[24]:
```
dt_classifier = DecisionTreeClassifier(random_state=598)
dt_classifier.fit(X_train, y_train)
y_pred_dt_train, y_pred_dt_test = dt_classifier.predict(X_train), dt_classifier.
 ↪predict(X_test)
```

1.3.5 Part C

Choose two boosting algorithms among AdaBoost, Gradient Boosting, or XGBoost and fit a classifier on the training data set.

[25]:
```
ab_classifier = AdaBoostClassifier(random_state=598)
ab_classifier.fit(X_train, y_train)
```

```
y_pred_ab_train, y_pred_ab_test = ab_classifier.predict(X_train), ab_classifier.
 ↪predict(X_test)
```

[26]:
```
gb_classifier = GradientBoostingClassifier(random_state=598)
gb_classifier.fit(X_train, y_train)
y_pred_gb_train, y_pred_gb_test = gb_classifier.predict(X_train), gb_classifier.
 ↪predict(X_test)
```

1.3.6 Part D

Report the accuracy for all classifiers fitted in (a), (b), (c) in both training and testing data

[28]:
```
print("SVM Classifier")
print("Train: ", accuracy_score(y_train, y_pred_svm_train))
print("Test: ", accuracy_score(y_test, y_pred_svm_test))

print("Decision Tree Classifier")
print("Train: ", accuracy_score(y_train, y_pred_dt_train))
print("Test: ", accuracy_score(y_test, y_pred_dt_test))

print("Adaboost Classifier")
print("Train: ", accuracy_score(y_train, y_pred_ab_train))
print("Test: ", accuracy_score(y_test, y_pred_ab_test))

print("Gradient Boost Classifier")
print("Train: ", accuracy_score(y_train, y_pred_gb_train))
print("Test: ", accuracy_score(y_test, y_pred_gb_test))
```

```
SVM Classifier
Train:  0.9966639978649586
Test:  0.9817038307604345
Decision Tree Classifier
Train:  1.0
Test:  0.9188107489994283
Adaboost Classifier
Train:  0.64758473445423
Test:  0.6097770154373928
Gradient Boost Classifier
Train:  1.0
Test:  0.9625500285877644
```

1.3.7 Part E

Report the confusion matrix for the test predictions.

[29]:
```
cm_svm_test = confusion_matrix(y_test, y_pred_svm_test)
cm_dt_test = confusion_matrix(y_test, y_pred_dt_test)
```

```
cm_ab_test = confusion_matrix(y_test, y_pred_ab_test)
cm_gb_test = confusion_matrix(y_test, y_pred_gb_test)
```

[30]:
```
print("SVM Confusion Matrix:\n", cm_svm_test)
print("DecisionTree Confusion Matrix:\n", cm_dt_test)
print("Adaboost Confusion Matrix:\n", cm_ab_test)
print("Gradient Boost Confusion Matrix:\n", cm_gb_test)
```

```
SVM Confusion Matrix:
 [[353   0   0   0   0   0   0   0  10   0]
 [  0 350  13   0   1   0   0   0   0   0]
 [  0   2 362   0   0   0   0   0   0   0]
 [  0   1   0 333   0   0   0   0   0   2]
 [  0   0   0   0 359   4   1   0   0   0]
 [  0   0   0   4   0 329   0   0   0   2]
 [  0   0   0   0   0   0 336   0   0   0]
 [  0  12   1   0   0   0   0 347   0   4]
 [  0   0   0   0   0   1   0   0 335   0]
 [  0   2   0   0   0   0   0   3   1 330]]
DecisionTree Confusion Matrix:
 [[344   0   1   0   1   0   1   1  13   2]
 [  0 318  41   1   1   1   1   1   0   0]
 [  0  10 351   0   0   0   1   2   0   0]
 [  1  11   2 314   0   1   0   3   0   4]
 [  0   2   0   1 354   2   4   1   0   0]
 [  0   3   0  25   2 287   0   1   5  12]
 [  4   4   1   0   7   4 315   0   1   0]
 [  0  39   4   8   0   0   1 308   4   0]
 [  5   1   0   1   5   3   6   4 311   0]
 [  0   6   0   4   5   3   0   3   3 312]]
Adaboost Confusion Matrix:
 [[337   0   1   0   0   0  23   0   2   0]
 [  0 212 138   5   4   0   2   0   0   3]
 [  0   8 334   1   0   0  13   8   0   0]
 [  0   2  60 245   0  19   9   0   0   1]
 [  1   9   3   0 325   0  19   0   0   7]
 [167  12   0 119   2  11  15   0   1   8]
 [  0   0   0   8   0   0 308   5  13   2]
 [  0  31  12   2   0   0   9 270  18  22]
 [266   0  10   0   0   0  13  28  19   0]
 [  1  24   9 168  12  30  20   0   0  72]]
Gradient Boost Confusion Matrix:
 [[343   0   0   0   0   0   0   0  20   0]
 [  0 340  22   0   1   1   0   0   0   0]
 [  0   3 360   0   0   0   0   1   0   0]
 [  0   3   0 331   0   0   0   1   0   1]
 [  0   1   0   0 363   0   0   0   0   0]
 [  0   0   0   5   0 311   0   0   4  15]
 [  0   0   0   0   0   1 334   0   1   0]
 [  0  19   3   0   0   0   0 327   0  15]
 [  0   0   0   0   0   0   0   0 336   0]
 [  0   6   0   6   0   0   0   1   1 322]]
```

1.3.8 Part F

Comment on the results. Which are the digits that seem to be most commonly confused? Did you have any overfitting issues with any of the approaches?

As can be seen through our results, it seems that the models that performed the best on the test data was the Gradient Boosting classifier and the SVM classifier. Adaboost performed poorly on both train and test predictions. The digits that saw the most issues with accuracy across all classifier models were 0, 5, and 7. 0 had a lot of predictions predict it as the digit 8. Similarly, 5 was confused with 3 and 7 with 1. Adaboost obviously did not perform well on these digits either (as seen from the results), but it additionally had issues with 1, 3, and 8. In terms of overfitting, not much issues were encountered.