# Winery classification with the Gaussian generative approach

December 11, 2019

## 1 Experimentations with Gaussian classification on the Winery data

This notebook will experiment with the Gaussian classification of the winery data, writing the code from scratch. It will use at univariate, bivariate and multivariate Gaussian.

The data is from the UCI repository (https://archive.ics.uci.edu/ml/datasets/wine).

The data contains 178 labeled data points, each corresponding to a bottle of wine: * The features (x): a 13-dimensional vector consisting of features for the bottle of wine (features are: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, Proline * The label (y): the winery where the bottle came from (1,2,3)

### 1.1 Load the data

[1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm, multivariate_normal
import ipywidgets as widgets
from IPython.display import display
from ipywidgets import interact, interactive, fixed, interact_manual, IntSlider
```

[2]:
```
data = np.loadtxt('wine.data.txt', delimiter=',')

featurenames = ['Alcohol', 'Malic acid', 'Ash', 'Alcalinity of␣
↪ash','Magnesium', 'Total phenols',
                'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color␣
↪intensity', 'Hue',
                'OD280/OD315 of diluted wines', 'Proline']
```

[3]:
```
# Split the data training set (trainx, trainy) of size 130 and test set (testx,␣
↪testy) of size 48
np.random.seed(0)
perm = np.random.permutation(178)
trainx = data[perm[0:130],1:14]
```

```
trainy = data[perm[0:130],0]
testx = data[perm[130:178], 1:14]
testy = data[perm[130:178],0]
```

[4]:
```
#number of each label in the train data
sum(trainy==1), sum(trainy==2), sum(trainy==3)
```

[4]: (43, 54, 33)

[5]:
```
#number of each label in test data
sum(testy==1), sum(testy==2), sum(testy==3)
```

[5]: (16, 17, 15)

### 1.2 Distributions of each feature for each winery

A histogram will be created for each feature under each class (winery), along with the Gaussian fit to the distribution. A slider functionality will be used to scroll through classes and features.

[6]:
```
@interact_manual( feature=IntSlider(0,0,12), label=IntSlider(1,1,3))
def density_plot(feature, label):
    plt.hist(trainx[trainy==label,feature], density=True)
    #
    mu = np.mean(trainx[trainy==label,feature]) # mean
    var = np.var(trainx[trainy==label,feature]) # variance
    std = np.sqrt(var) # standard deviation
    #
    x_axis = np.linspace(mu - 3*std, mu + 3*std, 1000)
    plt.plot(x_axis, norm.pdf(x_axis,mu,std), 'r', lw=2)
    plt.title("Winery "+str(label) )
    plt.xlabel(featurenames[feature], fontsize=14, color='red')
    plt.ylabel('Density', fontsize=14, color='red')
    plt.show()
```

```
interactive(children=(IntSlider(value=0, description='feature', max=12), IntSlider(value=1, des
```

## 2 Univariate

### 2.1 Fit a Gaussian generative model to each class for each feature individually

[7]:
```
# function to fit generative model for 3 classes for a single feature
def fit_generative_model(x,y,feature):
    k = 3 # number of classes
    mu = np.zeros(k+1) # list of means
    var = np.zeros(k+1) # list of variances
    pi = np.zeros(k+1) # list of class weights
```

```
    for label in range(1,k+1):
        indices = (y==label)
        mu[label] = np.mean(x[indices,feature])
        var[label] = np.var(x[indices,feature])
        pi[label] = float(sum(indices))/float(len(y))
    return mu, var, pi
```

[8]:
```
#display Gaussian distribution for each of the classes on the same plot
@interact_manual( feature=IntSlider(0,0,12) )
def show_densities(feature):
    mu, var, pi = fit_generative_model(trainx, trainy, feature)
    colors = ['r', 'k', 'g']
    for label in range(1,4):
        m = mu[label]
        s = np.sqrt(var[label])
        x_axis = np.linspace(m - 3*s, m+3*s, 1000)
        plt.plot(x_axis, norm.pdf(x_axis,m,s), colors[label-1], label="class "␣
↪+ str(label))
    plt.xlabel(featurenames[feature], fontsize=14, color='red')
    plt.ylabel('Density', fontsize=14, color='red')
    plt.legend()
    plt.show()
```

```
interactive(children=(IntSlider(value=0, description='feature', max=12), Button(description='Ru
```

### 2.2 Training error for each feature

How many classes are incorrectly predicted based on the Gaussian model for each feature individually?

[9]:
```
@interact( feature=IntSlider(0,0,12) )
def train_model(feature):
    mu, var, pi = fit_generative_model(trainx, trainy, feature)

    k = 3
    n_train = len(trainy)
    score = np.zeros((n_train,k+1))
    for i in range(0,n_train):
        for label in range(1,k+1):
            score[i,label] = np.log(pi[label]) + \
            norm.logpdf(trainx[i,feature], mu[label], np.sqrt(var[label]))
    predictions = np.argmax(score[:,1:4], axis=1) + 1
    errors = np.sum(predictions != trainy)
    print("Train error using feature " + featurenames[feature] + ": " +␣
↪str(errors) + "/" + str(n_train))
```

```
interactive(children=(IntSlider(value=0, description='feature', max=12), Output()), _dom_classe
```

### 2.3 Predicting labels for the test set using each feature individually

[10]:
```
@interact( feature=IntSlider(0,0,12) )
def test_model(feature):
    mu, var, pi = fit_generative_model(trainx, trainy, feature)

    k = 3
    n_test = len(testy)
    score = np.zeros((n_test,k+1))
    for i in range(0,n_test):
        for label in range(1,k+1):
            score[i,label] = np.log(pi[label]) + \
            norm.logpdf(testx[i,feature], mu[label], np.sqrt(var[label]))
    predictions = np.argmax(score[:,1:4], axis=1) + 1
    errors = np.sum(predictions != testy)
    print("Test error using feature " + featurenames[feature] + ": " +␣
 ↪str(errors) + "/" + str(n_test))
```

```
interactive(children=(IntSlider(value=0, description='feature', max=12), Output()), _dom_classe
```

[11]:
```
def train_error(feature):
    mu, var, pi = fit_generative_model(trainx, trainy, feature)

    k = 3
    n_train = len(trainy)
    score = np.zeros((n_train,k+1))
    train_error = []
    for i in range(0,n_train):
        for label in range(1,k+1):
            score[i,label] = np.log(pi[label]) + \
            norm.logpdf(trainx[i,feature], mu[label], np.sqrt(var[label]))
    predictions = np.argmax(score[:,1:4], axis=1) + 1
    errors = np.sum(predictions != trainy)
    return errors/n_train

def test_error(feature):
    mu, var, pi = fit_generative_model(trainx, trainy, feature)

    k = 3
    n_test = len(testy)
    score = np.zeros((n_test,k+1))
    for i in range(0,n_test):
        for label in range(1,k+1):
            score[i,label] = np.log(pi[label]) + \
            norm.logpdf(testx[i,feature], mu[label], np.sqrt(var[label]))
```

```
    predictions = np.argmax(score[:,1:4], axis=1) + 1
    errors = np.sum(predictions != testy)
    return errors/n_test

num = len(featurenames)
train_errors = []
for i in range(num):
    errors = train_error(i)
    train_errors.append(errors)

test_errors = []
for i in range(num):
    errors = test_error(i)
    test_errors.append(errors)

df = pd.DataFrame({'Feature': featurenames, 'Train Error': train_errors, 'Test␣
 ↪Error': test_errors})
df
```

[11]:

| | Feature | Train Error | Test Error |
|---|---|---|---|
| 0 | Alcohol | 0.338462 | 0.354167 |
| 1 | Malic acid | 0.376923 | 0.479167 |
| 2 | Ash | 0.507692 | 0.604167 |
| 3 | Alcalinity of ash | 0.523077 | 0.479167 |
| 4 | Magnesium | 0.469231 | 0.437500 |
| 5 | Total phenols | 0.353846 | 0.333333 |
| 6 | Flavanoids | 0.207692 | 0.166667 |
| 7 | Nonflavanoid phenols | 0.423077 | 0.479167 |
| 8 | Proanthocyanins | 0.461538 | 0.333333 |
| 9 | Color intensity | 0.292308 | 0.208333 |
| 10 | Hue | 0.369231 | 0.291667 |
| 11 | OD280/OD315 of diluted wines | 0.361538 | 0.395833 |
| 12 | Proline | 0.269231 | 0.354167 |

# 3 Bivariate

## 3.1 Distribution of two features from one winery

[12]:
```
#helper functions

# Function to fit a Gaussian to a data set using the selected features
def fit_gaussian(x, features):
    mu = np.mean(x[:,features], axis=0)
    covar = np.cov(x[:,features], rowvar=0, bias=1)
    return mu, covar
```

```
# Find the range within which an array of numbers lie, with a little buffer
def find_range(x):
    lower = min(x)
    upper = max(x)
    width = upper - lower
    lower = lower - 0.2 * width
    upper = upper + 0.2 * width
    return lower, upper

#function to plot a few contour lines of a given 2D Gaussian
def plot_contours(mu, cov, x1g, x2g, col):
    rv = multivariate_normal(mean=mu, cov=cov)
    z = np.zeros((len(x1g),len(x2g)))
    for i in range(0,len(x1g)):
        for j in range(0,len(x2g)):
            z[j,i] = rv.logpdf([x1g[i], x2g[j]])
    sign, logdet = np.linalg.slogdet(cov)
    normalizer = -0.5 * (2 * np.log(6.28) + sign * logdet)
    for offset in range(1,4):
        plt.contour(x1g,x2g,z, levels=[normalizer - offset], colors=col,␣
 ↪linewidths=2.0, linestyles='solid')
```

[13]:
```
#create function to visualise the distribution for individual wineries for a␣
 ↪pair of features
#using slider functionality
@interact_manual( f1=IntSlider(0,0,12,1), f2=IntSlider(6,0,12,1),␣
 ↪label=IntSlider(1,1,3,1) )
def two_features_plot(f1,f2,label):
    if f1 == f2: # we need f1 != f2
        print("Please choose different features for f1 and f2.")
        return

    # Set up plot
    x1_lower, x1_upper = find_range(trainx[trainy==label,f1])
    x2_lower, x2_upper = find_range(trainx[trainy==label,f2])
    plt.xlim(x1_lower, x1_upper) # limit along x1-axis
    plt.ylim(x2_lower, x2_upper) # limit along x2-axis

    # Plot training points along the two selected features
    plt.plot(trainx[trainy==label, f1], trainx[trainy==label, f2], 'ro')

    # Define a grid along each axis; the density will be computed at each grid␣
 ↪point
    res = 200 # resolution
    x1g = np.linspace(x1_lower, x1_upper, res)
    x2g = np.linspace(x2_lower, x2_upper, res)
```

```
    # plot a few contour lines of the density
    mu, cov = fit_gaussian(trainx[trainy==label,:], [f1,f2])
    plot_contours(mu, cov, x1g, x2g, 'k')

    plt.xlabel(featurenames[f1], fontsize=14, color='red')
    plt.ylabel(featurenames[f2], fontsize=14, color='red')
    plt.title('Class ' + str(label), fontsize=14, color='blue')
    plt.show()
```

```
interactive(children=(IntSlider(value=0, description='f1', max=12), IntSlider(value=6, descript
```

### 3.2 Fit a Gaussian generative model to each class for a pair of features

[14]:
```
# function to fit generative model based on 2 features
def fit_generative_model(x, y, features):
    k = 3 #classes
    d = len(features) # number of features
    mu = np.zeros((k+1,d)) # list of means
    covar = np.zeros((k+1,d,d)) # list of covariance matrices
    pi = np.zeros(k+1) # list of class weights
    for label in range(1,k+1):
        indices = (y==label)
        mu[label,:], covar[label,:,:] = fit_gaussian(x[indices,:], features)
        pi[label] = float(sum(indices))/float(len(y))
    return mu, covar, pi
```

[15]:
```
#plot the Gaussians for all 3 wineries for a pair of features
#using slider functionality to change pair of features
@interact_manual( f1=IntSlider(0,0,12,1), f2=IntSlider(6,0,12,1) )
def three_class_plot(f1,f2):
    if f1 == f2: # we need f1 != f2
        print("Please choose different features for f1 and f2.")
        return

    # Set up plot
    x1_lower, x1_upper = find_range(trainx[:,f1])
    x2_lower, x2_upper = find_range(trainx[:,f2])
    plt.xlim(x1_lower, x1_upper) # limit along x1-axis
    plt.ylim(x2_lower, x2_upper) # limit along x2-axis

    # Plot the training points along the two selected features
    colors = ['r', 'k', 'g']
    for label in range(1,4):
        plt.plot(trainx[trainy==label,f1], trainx[trainy==label,f2],␣
 ↪marker='o', ls='None', c=colors[label-1])
```

```
    # Define a grid along each axis; the density will be computed at each grid␣
↪point
    res = 200 # resolution
    x1g = np.linspace(x1_lower, x1_upper, res)
    x2g = np.linspace(x2_lower, x2_upper, res)

    # Show the Gaussian fit to each class, using features f1,f2
    mu, covar, pi = fit_generative_model(trainx, trainy, [f1,f2])
    for label in range(1,4):
        gmean = mu[label,:]
        gcov = covar[label,:,:]
        plot_contours(gmean, gcov, x1g, x2g, colors[label-1])

    plt.xlabel(featurenames[f1], fontsize=14, color='red')
    plt.ylabel(featurenames[f2], fontsize=14, color='red')
    plt.title('Wine data', fontsize=14, color='blue')
    plt.show()
```

```
interactive(children=(IntSlider(value=0, description='f1', max=12), IntSlider(value=6, descript
```

### 3.3 Predict labels for test point

[16]:
```
# test performance of predictor based on a subset of features
@interact( f1=IntSlider(0,0,12,1), f2=IntSlider(6,0,12,1) )
def test_model(f1, f2):
    if f1 == f2: # need f1 != f2
        print("Please choose different features for f1 and f2.")
        return
    features= [f1,f2]
    mu, covar, pi = fit_generative_model(trainx, trainy, features)

    k = 3
    nt = len(testy) # Number of test points
    score = np.zeros((nt,k+1))
    for i in range(0,nt):
        for label in range(1,k+1):
            score[i,label] = np.log(pi[label]) + \
            multivariate_normal.logpdf(testx[i,features], mean=mu[label,:],␣
↪cov=covar[label,:,:])
    predictions = np.argmax(score[:,1:4], axis=1) + 1
    # Finally, tally up score
    errors = np.sum(predictions != testy)
    print("Test error using feature(s): ",)
    for f in features:
        print("'" + featurenames[f] + "'" + " ",)
    print
```

```
    print("Errors: " + str(errors) + "/" + str(nt))
```

```
interactive(children=(IntSlider(value=0, description='f1', max=12), IntSlider(value=6, descript
```

### 3.4 Decision Boundary

A function will now be created that plots the decision boundary between the 3 wineries based on a specified pair of features.

[17]:
```
@interact_manual( f1=IntSlider(0,0,12,1), f2=IntSlider(6,0,12,1) )
def show_decision_boundary(f1,f2):
    # Fit Gaussian to each class
    mu, covar, pi = fit_generative_model(trainx, trainy, [f1,f2])

    # Set up dimensions of plot
    x1_lower, x1_upper = find_range(trainx[:,f1])
    x2_lower, x2_upper = find_range(trainx[:,f2])
    plt.xlim([x1_lower,x1_upper])
    plt.ylim([x2_lower,x2_upper])

    # Plot points in training set
    colors = ['r', 'k', 'g']
    for label in range(1,4):
        plt.plot(trainx[trainy==label,f1], trainx[trainy==label,f2],␣
↪marker='o', ls='None', c=colors[label-1])

    # Define a dense grid; every point in the grid will be classified according␣
↪to the generative model
    res = 200
    x1g = np.linspace(x1_lower, x1_upper, res)
    x2g = np.linspace(x2_lower, x2_upper, res)

    # Declare random variables corresponding to each class density
    random_vars = {}
    for label in range(1,4):
        random_vars[label] = multivariate_normal(mean=mu[label,:
↪],cov=covar[label,:,:])

    # Classify every point in the grid; these are stored in an array Z[]
    Z = np.zeros((len(x1g), len(x2g)))
    for i in range(0,len(x1g)):
        for j in range(0,len(x2g)):
            scores = []
            for label in range(1,4):
                scores.append(np.log(pi[label]) + random_vars[label].
↪logpdf([x1g[i],x2g[j]]))
```

```
            Z[i,j] = np.argmax(scores) + 1

    # Plot the contour lines
    plt.contour(x1g,x2g,Z.T,3,cmap='seismic')

    # Finally, show the image
    plt.xlabel(featurenames[f1], fontsize=14, color='red')
    plt.ylabel(featurenames[f2], fontsize=14, color='red')
    plt.show()
```

```
interactive(children=(IntSlider(value=0, description='f1', max=12), IntSlider(value=6, descrip
```

[18]:
```
#the same function as above will be used to visualise the decision boundaries␣
 ↪found from the training data
#in relation to the test data

@interact_manual( f1=IntSlider(0,0,12,1), f2=IntSlider(6,0,12,1) )
def show_decision_boundary(f1,f2):
    # Fit Gaussian to each class
    mu, covar, pi = fit_generative_model(trainx, trainy, [f1,f2])

    # Set up dimensions of plot
    x1_lower, x1_upper = find_range(testx[:,f1])
    x2_lower, x2_upper = find_range(testx[:,f2])
    plt.xlim([x1_lower,x1_upper])
    plt.ylim([x2_lower,x2_upper])

    # Plot points in training set
    colors = ['r', 'k', 'g']
    for label in range(1,4):
        plt.plot(testx[testy==label,f1], testx[testy==label,f2], marker='o',␣
 ↪ls='None', c=colors[label-1])

    # Define a dense grid; every point in the grid will be classified according␣
 ↪to the generative model
    res = 200
    x1g = np.linspace(x1_lower, x1_upper, res)
    x2g = np.linspace(x2_lower, x2_upper, res)

    # Declare random variables corresponding to each class density
    random_vars = {}
    for label in range(1,4):
        random_vars[label] = multivariate_normal(mean=mu[label,:
 ↪],cov=covar[label,:,:])

    # Classify every point in the grid; these are stored in an array Z[]
```

```
    Z = np.zeros((len(x1g), len(x2g)))
    for i in range(0,len(x1g)):
        for j in range(0,len(x2g)):
            scores = []
            for label in range(1,4):
                scores.append(np.log(pi[label]) + random_vars[label].
↪logpdf([x1g[i],x2g[j]]))
            Z[i,j] = np.argmax(scores) + 1

    # Plot the contour lines
    plt.contour(x1g,x2g,Z.T,3,cmap='seismic')

    # Finally, show the image
    plt.xlabel(featurenames[f1], fontsize=14, color='red')
    plt.ylabel(featurenames[f2], fontsize=14, color='red')
    plt.show()
```

```
interactive(children=(IntSlider(value=0, description='f1', max=12), IntSlider(value=6, descript
```

# 4 Multivariate

[19]:
```
#function that uses all features to create generative model
def fit_generative_model(x,y):
    k = 3  # labels 1,2,...,k
    d = (x.shape)[1]  # number of features
    mu = np.zeros((k+1,d))
    sigma = np.zeros((k+1,d,d))
    pi = np.zeros(k+1)
    for label in range(1,k+1):
        indices = (y == label)
        mu[label] = np.mean(x[indices,:], axis=0)
        sigma[label] = np.cov(x[indices,:], rowvar=0, bias=1)
        pi[label] = float(sum(indices))/float(len(y))
    return mu, sigma, pi
```

[20]:
```
# Fit a Gaussian generative model to the training data
mu, sigma, pi = fit_generative_model(trainx,trainy)
```

## 4.1 Make predictions on the test set

A function will be created to indicate test error, where any number of features to be inputted can be specified.

[21]:
```
# test the performance of a predictor based on specified features
def test_model(mu, sigma, pi, features, tx, ty):
    k = 3
```

```
    nt = len(ty)
    score = np.zeros((nt,k+1))
    tx = tx[:,[features]]
    for i in range(0,nt):
        for label in range(1,k+1):
            aa = sigma[label,features,:]
            aa1 = aa[:,features]
            score[i,label] = np.log(pi[label]) + \
            multivariate_normal.logpdf(tx[i], mean=mu[label,features], cov=aa1)
        predictions = np.argmax(score[:,1:4], axis=1) + 1

    errors = np.sum(predictions != ty)
    print("Test error " + ": " + str(errors) + "/" + str(nt))
```

[22]:
```
#test error with a single feature
test_model(mu, sigma, pi, [2], testx, testy)
```

```
Test error : 29/48
```

[23]:
```
#test error using 2 features
test_model(mu, sigma, pi, [0,2], testx, testy)
```

```
Test error : 12/48
```

[24]:
```
#test error using 3 features
test_model(mu, sigma, pi, [2,4,6], testx, testy)
```

```
Test error : 7/48
```

[25]:
```
#test error using all features
test_model(mu, sigma, pi, range(0,13), testx, testy)
```

```
Test error : 2/48
```

[ ]:
```

```