Iris K-Nearest Neighbors

Sid Shapiro
Nick McClellan

April 22, 2021

Abstract

We analyzed the Iris data using a k-nearest neighbor classifier. We scaled the data using the mean and standard deviation, then visualized it using Principal Components Analysis. Next we determined an optimal value for the number of nearest neighbors using a 5-fold cross validation study. Finally, we built our classifier and reported the results. We used 8 nearest neighbors, and found an accuracy of 100% using our technique.

1 Problem Discussion

We looked at a data set containing three different types of iris flowers, Setosa, Versicolour, and Virginica. It was introduced by British statistician and biologist Ronald Fisher in 1936. Our data consisted of fifty samples from each type of iris named above, and each sample consists of four measurements of the flower. These four measurements were the length of the pedal, the width of the pedal, the length of the sepal, and the width of the flowers sepal.[1]

In the next section, we give a brief overview of the data and scale the data appropriately. The following section we build the nearest neighbors classifier. Finally, we take a look at the results of our model.

2 The Data

The data consists of four possible variables and 150 samples of flowers. So the original data matrix is 4×150. Before we visualize the data, the four variables are:

1. Length of the pedal
2. Width of the pedal
3. Length of the sepal
4. Width of the sepal

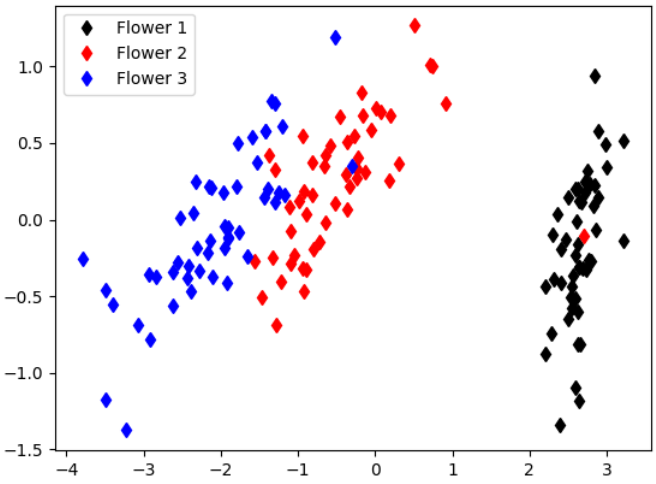
The mean and standard deviation of the 8 variables are given in the table below.

	L pedal	W pedal	L sepal	W sepal
mean	5.843	3.057	3.758	1.199
std	0.825	0.434	1.759	0.759

There was enough differences in the scales that we decided to perform a standard scaling across all variables to make each have zero mean and unit standard deviation.

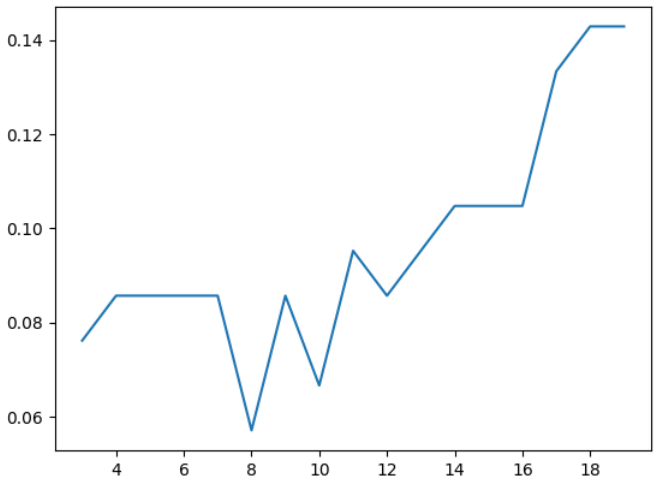
To visualize the three classes, we performed a Principal Components Analysis on the data, with the results shown graphically. The black diamonds are class 1, the red diamonds are class 2, and the blue diamonds are class 3.

Before going into the classification stage, we reserved 30% of the data (45 points) chosen randomly, for the test set, and that left us with 105 points for training.



3 Construction of the Classifier

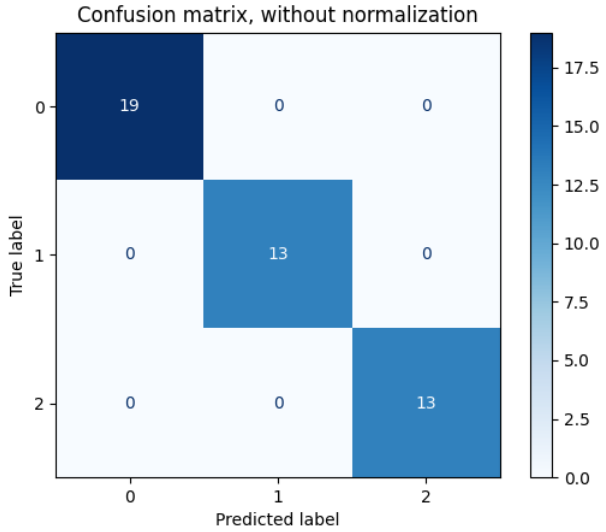
We performed a 5-fold cross validation study to determine the best number of neighbors. Repeating the process several times led us to determine an approximate number of neighbors, 8. The graph below shows the error rate for each choice of neighbors, from 3 to 20.



Finally, we used the full training set to classify the test set that was reserved initially.

4 Results

The results of the classification are summarized below in the confusion matrix. The predicted classes are shown in rows, the actual classes in the columns.



We had pretty close to an equal class representation in our randomized test data, as you can see all three flower types had thirteen to nineteen flowers. We can had an astounding amount of accuracy in our model, as you can see none of the 45 test flowers were classified as the wrong flower. Therefore, the overall accuracy was 100%.

References

[1] Wikipedia Foundation (2021). Iris flower data set. Retrieved at https://en.wikipedia.org/wiki/Iris_flower_data_set

5 Appendix: Code

```
import matplotlib.pyplot as plt

import numpy as np
from numpy import mean
from numpy import std

import scipy.io
from scipy.stats import sem

import sklearn
from sklearn.model_selection import KFold, train_test_split, cross_val_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

#Function Definition for use later
def evaluate_model(X, t, k):
    cv = KFold(n_splits=5)
# create model
    model = KNeighborsClassifier(n_neighbors=k)
# evaluate model
```

```
    scores = cross_val_score(model, X, t, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

#Get data from iris dataset-----------------------------------------------
from sklearn import datasets
iris = datasets.load_iris()
X = np.array(iris.data)
T = np.array(iris.target)

#Visualize Data with PCA
Xm= (X-np.mean(X,axis=0)).T
U, S, VT = np.linalg.svd(Xm,full_matrices=0)
L=U[:,0:2].T @ Xm

plt.figure()
plt.plot(L[0,:49],L[1,:49],'d',color='k',label='Flower 1')
plt.plot(L[0,49:99],L[1,49:99],'d',color='r',label='Flower 2')
plt.plot(L[0,99:149],L[1,99:149],'d',color='b',label='Flower 3')
plt.legend()
plt.show()

#Building the classifier---------------------------------------------------
#We will scale the data first by mean subtracting and dividing by the standard deviation of X
Sd = np.std(X,axis=0)
Xs=Xm/Sd.T[:,np.newaxis]


#Allocate Testing and Training Data
X_train, X_test, T_train, T_test = train_test_split(Xs.T,T,test_size=.3,random_state=42)

# Cross Validates data with varying number of Neighboors
neighbors = range(3,20)
results = list()

for r in neighbors:
# evaluate using a given number of repeats
scores = evaluate_model(X_train, T_train, r)
# summarize
print('>%d mean=%.4f se=%.3f' % (r, mean(scores), sem(scores)))
# store
results.append(1-mean(scores))

plt.figure()
plt.plot(neighbors,results)
plt.show()


# Final Model and confusion matrix:
classifier = KNeighborsClassifier(n_neighbors=8).fit(X_train, T_train)

np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(classifier, X_test, T_test,
```

```
                                cmap=plt.cm.Blues,
                                normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```