Classification

Practical 6

k-NN
Q1

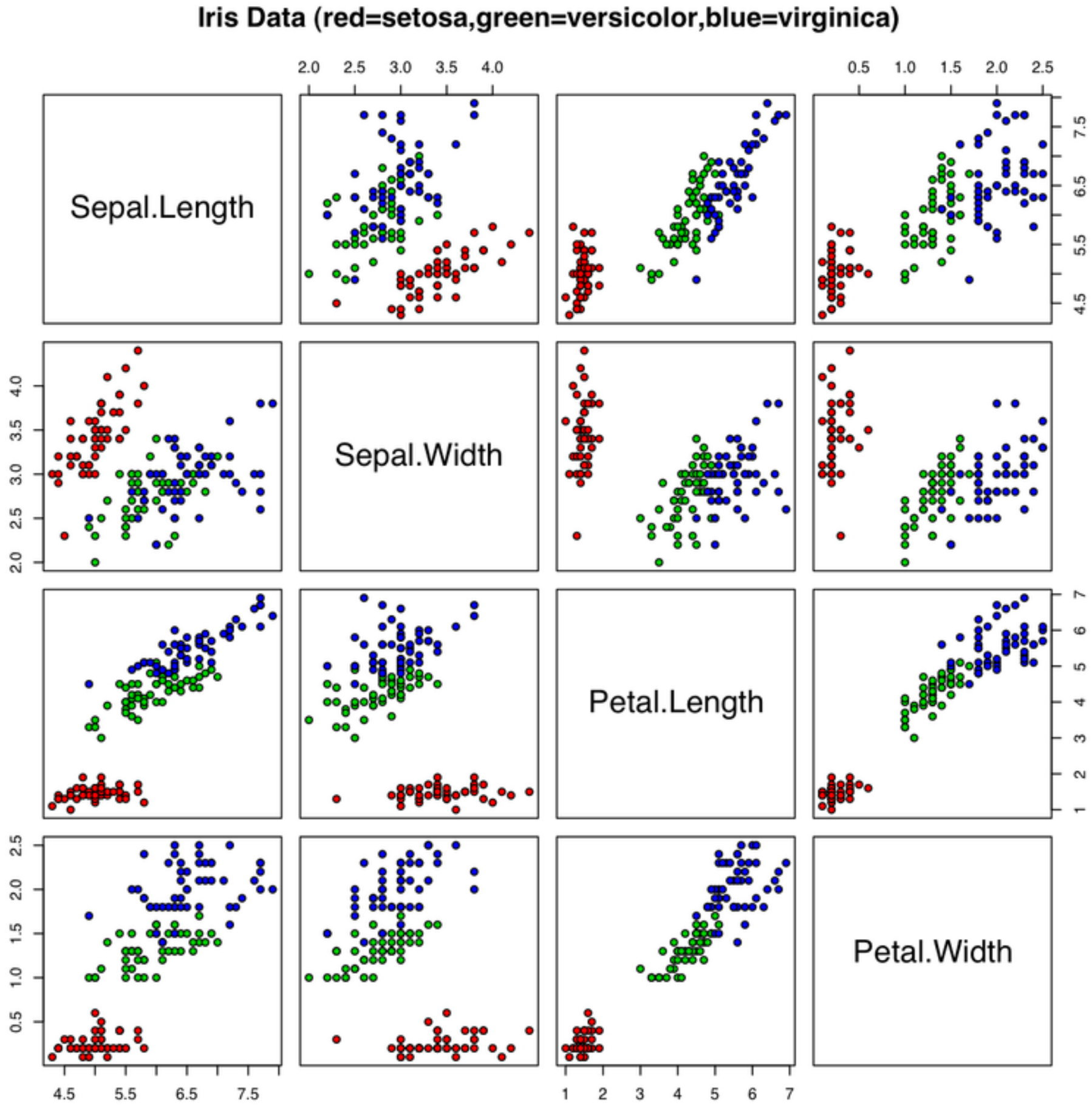
You have lovely irises….(Paul Durkan)

Classify Flowers Using Measurements

The test problem we will be using in this tutorial is iris classification.

The problem is comprised of 150 observations of iris flowers from three different species. There are 4 measurements of given flowers: sepal length, sepal width, petal length and petal width, all in the same unit of centimeters. The predicted attribute is the species, which is one of setosa, versicolor or virginica.

It is a standard dataset where the species is known for all instances. As such we can split the data into training and test datasets and use the results to evaluate our algorithm implementation. Good classification accuracy on this problem is above 90% correct, typically 96% or better.


Iris Data (red=setosa,green=versicolor,blue=virginica)
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width

```
# Example of kNN implemented from Scratch in Python
# By Jason Brownlee
#http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-py

import csv
import random
import math
import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]):
        with open(filename, 'r') as csvfile:
            lines = csv.reader(csvfile)
            dataset = list(lines)
            for x in range(len(dataset)-1):
                for y in range(4):
                    dataset[x][y] = float(dataset[x][y])
                if random.random() < split:
                    trainingSet.append(dataset[x])
                else:
                    testSet.append(dataset[x])


def euclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                distance += pow((instance1[x] - instance2[x]), 2)
        return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
        distances = []
        length = len(testInstance)-1
        for x in range(len(trainingSet)):
                dist = euclideanDistance(testInstance, trainingSet[x], length)
                distances.append((trainingSet[x], dist))
        distances.sort(key=operator.itemgetter(1))
        neighbors = []
        for x in range(k):
                neighbors.append(distances[x][0])
        return neighbors
```

```
def getNeighbors(trainingSet, testInstance, k):

def getResponse(neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][-1]
                if response in classVotes:
                        classVotes[response] += 1
                else:
                        classVotes[response] = 1
        sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reve
        return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
                if testSet[x][-1] == predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0

def main():
        # prepare data
        trainingSet=[]
        testSet=[]
        split = 0.50
        loadDataset('iris.csv', split, trainingSet, testSet)
        print('Train set: ' + repr(len(trainingSet)))
        print('Test set: ' + repr(len(testSet)))
        # generate predictions
        predictions=[]
        k = 2
        for x in range(len(testSet)):
                neighbors = getNeighbors(trainingSet, testSet[x], k)
                result = getResponse(neighbors)
                predictions.append(result)
                print('> predicted=' + repr(result) + ', actual=' + repr(testSet[
        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: ' + repr(accuracy) + '%')

main()
```

```
def getNeighbors(trainingSet, testInstance, k):

def getResponse(neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][-1]
                if response in classVotes:
                        classVotes[response] += 1
                else:
                        classVotes[response] = 1
        sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reve
        return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
                if testSet[x][-1] == predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0

def main():
        # prepare data
        trainingSet=[]
        testSet=[]
        split = 0.50
        loadDataset('iris.csv', split, trainingSet, testSet)
        print('Train set: ' + repr(len(trainingSet)))
        print('Test set: ' + repr(len(testSet)))
        # generate predictions
        predictions=[]
        k = 2
        for x in range(len(testSet)):
                neighbors = getNeighbors(trainingSet, testSet[x], k)
                result = getResponse(neighbors)
                predictions.append(result)
                print('> predicted=' + repr(result) + ', actual=' + repr(testSet[
        accuracy = getAccuracy(testSet, predictions)
        print('Accuracy: ' + repr(accuracy) + '%')

main()
```

Q1: k-NN

- Check out the provided program on k-NN, applied to the famous Iris Data set
- Two parameters are interesting:
  - the ***split*** which is the size of the training subset v test subset (split = .67) means roughly 2/3rds training, 1/3rd testing
  - ***k*** which is the size of nearest neighbours
- ***Accuracy*** of model is determined for test sets

Your Task I

- Taking ideas about cross-validation into account; you job is first to systematically vary the size of the ***split*** ; exploring a decent number of other values for it >0.0 and <0.9 (to be chosen by you)
- Also to systematically vary ***k*** on five selected values between 1 and 20
- Then plot accuracy in a graph for these parameter changes

Your Task II

- Now, using this data set, describe an algorithm for doing a 5-fold cross validation on this data set
- See can you implement it in Python

Naive Bayes
Q2

Prac6 Q2 Naive Bayes

- Have a look at the nltk Bayes classifier that does the prediction of male/female names based on the last letter in the name
- Think of a new feature that you could extract from the data-set; define a fn for it (modifying **`gender_features`**) and see what is learned from it in the classification
- Compare the accuracy of your feature versus that of the last_letter feature and discuss.

SVM
Q3

Prac6 Q3 SVM Prog

svmeg.py - /Users/user/Dropbox/Teaching.TextAnalytics/Lect6.Classifiers/xLect6.Progs/svmeg.py (3.4.3)

```
__author__ = 'user'
# http://pythonprogramming.net/support-vector-machine-svm-example-tutorial-scikit-

import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn import svm

digits = datasets.load_digits()

classifier = svm.SVC(gamma=0.01, C=100)

print(len(digits.data))

x, y = digits.data[:-1], digits.target[:-1]
classifier.fit(x, y)

print('Prediction:', classifier.predict(digits.data[-1]))

plt.imshow(digits.images[-1], cmap=plt.cm.gray_r, interpolation='nearest')
plt.show()
```

Prac6 Q3 SVM

- So, this is just a quick use of an SVM; to show you how easy it is
- It involves learning digit recognition
- NB to install packages: sklearn, matplotlib, scipy; (problems around installing scipy in on mac with Pycharm; but works with “port install py34-scipy”
- Now run three different training configurations and then test the outputs for 5-10 different digits; report results