CS168 Spring Assignment 4

SUNet ID(s): johngold jsalloum
Name(s): John Gold Justin Salloum
Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Part 1

(a) Warm-Up

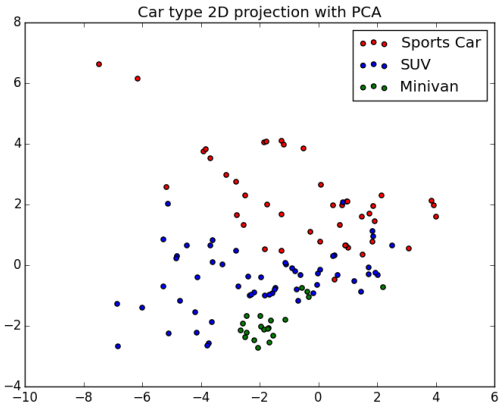
(b) Vector 1:[-0.27545832, -0.27368854, -0.34523251, -0.33333868, -0.31896962, 0.30775548, 0.30367002, -0.33514127, -0.26438741, -0.25027963, -0.2922094]

Vector 2:[0.44424397, 0.44625723, -0.01076804, 0.09509022, 0.27998005, -0.01300475, -0.02798239, -0.15767105, -0.43021634, -0.43979562, -0.33412014]

(c) Discussion: The two positive coordinates of v_0 refer to CityMPG and HighwayMPG (the 6th and 7th columns). This means that (1) CityMPG and HighwayMPG are closely correlated, and (2), their relationship is inverted compared to the other statistics. In general, as the other numbers increase, MPG will decrease, and vice versa. Cars with high projections onto v_0 should have high MPG ratings and be less expensive. The other values such as Cylinders, Horsepower, length and width should also be smaller in comparision to the average car.

Cars with low projections onto v_0 will have low MPG ratings and be more expensive in addition higher values for all other metrics (horsepower, weight, length, width...)

(d) Plot:

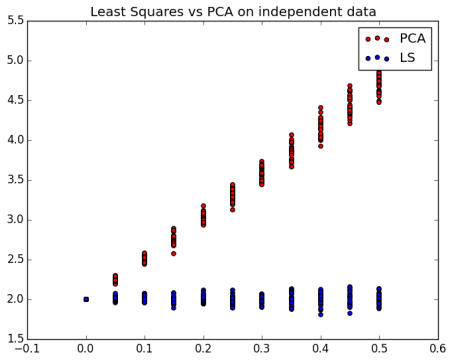


(e) Discussion: Minivans are clustered most strongly, generally around (-2, -2). This means minivans are very similar with all other types of minivans. Their MPG/price/size do not have much variance inside the minivan category. Sports cars and SUVs are not nearly as well clustered, as their are many different types, from cheap to luxury. There is not a market for "luxury minivans", though SUVs and Sports Cars have a wider range of appeal.

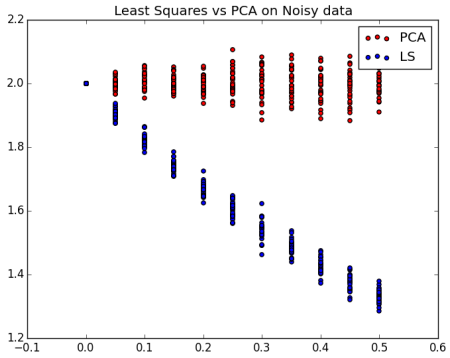
Part 2

(a) Warm-Up

(b) Plot (X is independent)



(c) Plot (Noisy estimates)



(d) Discussion: Least Squares does better on the independent X data, but PCA does better with the noisy data. As C varies in part 2b, Least Squares still maintains the relationship of Y=2X, even with the guassian randomness added. In part 2c, we have noisy estimates of both X and Y, so while PCA doesn't change as we vary C, Least Squares is fooled into believing there is a relationship.

Code Appendix

```
#p4.py
import random
import numpy as np
from numpy import linalg as LA
from numpy import mean, std, cov
from numpy.linalg import eigh
import pandas as pd
import matplotlib.pyplot as plt
import sys

def main():
    if(sys.argv[1] == "1"):
        df = pd.read_csv('p4_dataset.csv')
        car_categories = list(df['Category'])
        #car_models = list(df['Model'])

        #print car_categories
        raw_data = np.genfromtxt('p4_dataset.csv', delimiter=',', skip_header
        pca_results = PCA(raw_data)
        center_scaled_data = raw_data - mean(raw_data, 0)
        center_scaled_data /= std(center_scaled_data, 0)
        part_1d_graph(car_categories, pca_results, center_scaled_data)


    else:
        C = [.05 * i for i in range(11)]
        Y = None
        for trial in range(30):
            for c in C:
                if(sys.argv[1] == "2b"):
                    X = np.array([.001 * i for i in range(1,1001)])
                    Y = 2*X + np.random.randn(1000) * np.sqrt(c)
                    plt.title("Least Squares vs PCA on Independent data")
                else:
                    Y = [2*i/1000.0 + np.random.random() * np.sqrt(c) for i i
                    X = [i/1000.0 + np.random.random() * np.sqrt(c) for i in
                    plt.title("Least Squares vs PCA on Noisy data")

                #c on horizontal, output from recovery on vertical
                pca_recovered = plt.scatter([c], pca_recover(X,Y), c="r")
```

```
                ls_recovered = plt.scatter([c], ls_recover(X,Y), c="b")
                plt.legend([pca_recovered, ls_recovered], ("PCA", "LS"))
        plt.show()
    #elif(sys.argv[1] == "2a"):
    #    part_2a_graph()



#Uses the first 2 principle components to look at Sports cars, suvs, minivans
def part_1d_graph(car_categories, pca_results, raw_data):
    sports_x = []
    sports_y = []
    suv_x = []
    suv_y = []
    minivan_x = []
    minivan_y = []
    v_0 = pca_results[0]
    v_1 = pca_results[1]
    for index, car_type in enumerate(car_categories):
        if(car_type == "sports"):
            sports_x.append(v_0.dot(raw_data[index,:]))
            sports_y.append(v_1.dot(raw_data[index,:]))

        if(car_type == "minivan"):
            minivan_x.append(v_0.dot(raw_data[index,:]))
            minivan_y.append(v_1.dot(raw_data[index,:]))
        if(car_type == "suv"):
            suv_x.append(v_0.dot(raw_data[index,:]))
            suv_y.append(v_1.dot(raw_data[index,:]))

    sports = plt.scatter(sports_x, sports_y,c="r")
    suv = plt.scatter(suv_x, suv_y,c="b")
    minivan = plt.scatter(minivan_x, minivan_y, c="g")
    plt.title("Car type 2D projection with PCA")
    plt.legend([sports, suv, minivan], ("Sports Car", "SUV", "Minivan"))
    plt.show()



# data[0] is the first data element, etc
def PCA(data, normalize = True):
    data -= mean(data, 0)
```

```
    if normalize:
        data /= std(data, 0)
    C = cov(data, rowvar=0) # covariance matrix
    w,V = eigh(C) # w = eigenvalues, V[:,w] = corresponding eigenvectors
    # return the eigenvectors ordered by w (in decreasing order)
    return [V[:,i] for i,e in sorted(enumerate(w), key = lambda x: x[1], reve


# takes a vector X of xi's and a vector Y of yi's and returns the
# slope of the first component of the PCA (namely, the second coordinate divi
def pca_recover(X, Y):
    matrix = np.vstack((X,Y))
    pca = PCA(matrix.T, False)
    #hmmm...
    return pca[0][1]/pca[0][0]

#X and Y and returns the slope of the least squares fit.
#(Hint: since X is one dimensional, this takes a particularly simple form: X.
def ls_recover(X, Y):
    numerator = (X - np.mean(X)).dot(Y - np.mean(Y))
    denominator = LA.norm((X - np.mean(X)))**2
    return numerator/denominator

def test_recovery():
    X = np.array([.001 * i for i in range(1,1001)])
    Y = np.array([2*xi for xi in X])

    print pca_recover(X, Y)
    print ls_recover(X, Y)

if __name__ == "__main__":
    main()
    #test_recovery()
```