Statistical Machine Learning
Homework 3

Patrick Nowak
Rinor Cakaj

Sommersemester 2020
July 10, 2020
Sheet 3

Task 3.1: PCA

We use the data-set iris.txt. Each row in it has 4 Attributes and a label(classification).

3.1a)

We normalize the data, s.t. every Attribute has mean zero and variance 1. Therefore, for every column, we compute the sample mean and subtract it. Then, we devide by the standard deviation. Of course, this is only done for the columns containing attributes.

```
#3a
n = len(iris)
pre_iris = iris[:,0:4]
pred = iris[:,4]
mean = pre_iris.mean(0)
step1 = pre_iris - mean
standard_deviation=np.std(pre_iris, axis=0)
normalized_data = np.multiply(step1, 1/standard_deviation).T
```

”normalized_data” does now contain one row per attribute, each row having mean zero and variance one.

3.1b)

We apply PCA on the normalized dataset. We plot a bar graph where the i-th bar shows how much of the original variance we already captured using the biggest i components.

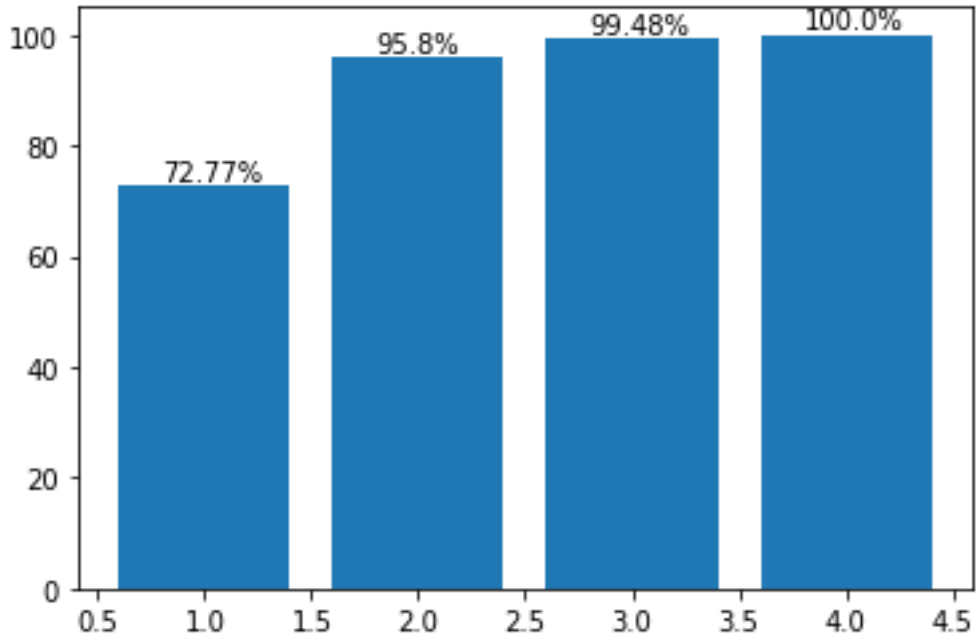
```
def PCA(normalized_data):
        cov = np.cov(normalized_data)
        eigenvalues, eigenvectors = np.linalg.eig(cov)

        summe = np.sum(eigenvalues)
        eigenvalues_prop = eigenvalues/summe

        ind = [1,2,3,4]
        kum = np.zeros(4)
        kum[0] = eigenvalues_prop[0]
        for i in range(1,4):
                kum[i]=kum[i-1] + eigenvalues_prop[i]
        kum=kum*100
        plt.bar(ind, kum)
        rounded=np.round(kum,2)
        for idx,y in enumerate(rounded):
```

```
    plt.text(ind[idx]-0.2,y+1,str(y)+"%")
    plt.show()
    return eigenvalues, eigenvectors
```

Calling the function produces the following plot:



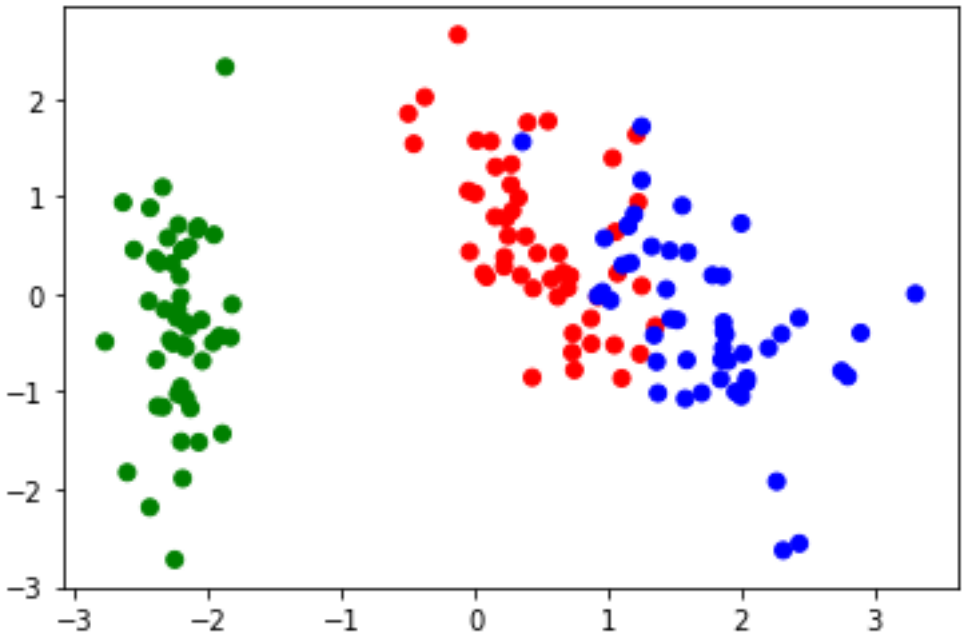
We see that for two components we explained already more than 95% of the variance.

3.1c)

We do the PCA for the two biggest components. We wrote the following function:

```
def threec(eigenvalues, eigenvectors, normalized_data, pred):
        B = eigenvectors[:,0:2]
        normalized_data_p = np.matmul(B.T, normalized_data)
        normalized_data_p = np.vstack((normalized_data_p, pred))
        colors = ['red','green','blue']
        plt.scatter(normalized_data_p[0], normalized_data_p[1],
                c=normalized_data_p[2], cmap=matplotlib.colors.ListedColormap(colors)
        return
```

The first input arguments are clear, in the last argument we give the real data values from iris.txt just without the classification column. We then used the first two eigenvalues for projection on the R^2. Calling it, we get



where we used different colors for different classes(red=Setosa,green=Versicolour,blue=Virginica). We observe, that the green dots are well separated from the rest. Therefore we can argue, that Versicolour is pretty unique compared to the other two. For the red and blue dots it can be harsh to find a good decision boundary around x=1 we got both blue and red samples. The Setosa and the Virginica data seem to have more in common. Though, if we go away from the boundary, we could use the points to get - at least on the given training data - a very clear classification.

3.1d)

We perform the PCA for n=1,2,3,4 components. Then we take the computed points from R^n and embed them in R^4 to calculate an error distance. Doing this, we need to consider the normalization we have on our normalized_data, which was, for real dataset X with mean $\bar{x}$ and variance std^2

$$normalized\_data = \frac{X - \bar{x}}{std}$$

Following the idea of slide 27 in lecture 10, we get

$$\begin{aligned} a^n &= B^T \cdot (X - \bar{x}) \\ &= B^T \cdot (normalized\_data \cdot std) \end{aligned}$$

$$\tilde{x}^n = \bar{x} + B \cdot a^n$$

where B is the matrix of the first n eigenvalues. We implemented this backtransformation in the comp_set function.

```
def comp_set(n, eigenvectors, normalized_data, mean, var):
        B = eigenvectors[:,0:n+1]
        normalized_data_p = np.matmul(B.T, normalized_data)
        reconstruction = np.matmul(B, normalized_data_p)
        rec=np.multiply(reconstruction.T,var)
        rec=rec+mean
        return rec
```

Now, we can compare the backtransformed data to the original uncompressed data via normalized mean squared error.

```
def rmse(x):
        return np.sqrt(np.mean(x**2))

def threed(eigenvalues, eigenvectors, normalized_data, pre_iris, mean, var):
        lsg=np.zeros([4,4])
        for comp in range(4):
                re = comp_set(comp, eigenvectors, normalized_data, mean, var)
                diff = re - pre_iris
        for feature in range(4):
                lsg[comp,feature]=rmse(diff[:,feature])/
                        (np.sum(pre_iris[:,feature]/len(pre_iris)))
        print("lsg:", lsg)
        return
```

Calling the threed function gives us the following solution:

	x_1	x_2	x_3	x_4
n=1	6.406%	12,641%	6.021%	16.642%
n=2	3.945%	1.339%	5.946%	16.158%
n=3	0.531%	0.252%	5.381%	4.769%
n=4	0	0	0	0