Exercise Sheet 3 - Solutions for Theory Part

Group Name: EAJC

Bolkar EREN
Muhammet Ali AL
Maleyka JABBARLI
Aycan CELIK

Exercise 1

a) Given the first expression with the term $\boldsymbol{m}$, we will show an equality. Consider the dot product as the matrix multiplication of a row matrix of shape $1 \times d$ with a column matrix of shape $d \times 1$, thus, the result is the matrix of shape 1×1 which is invariant under the transpose. Then:

$$\begin{aligned}
\frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})]^2 &= \frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})][\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})] \\
&= \frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})][\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})]^\top \\
&= \frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})][(\boldsymbol{x}_i - \boldsymbol{m})^\top\boldsymbol{u}^{\top\top}] \\
&= \frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{m})^\top\boldsymbol{u}] \\
&= \boldsymbol{u}^\top\Big[\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{m})^\top\Big]\boldsymbol{u} \\
&= \boldsymbol{u}^\top\boldsymbol{\Sigma}\boldsymbol{u}
\end{aligned}$$

This deduction works vice-versa since there is a direct equation between the expressions whose argmax values are the values we are interested in.

Therefore,

$$\underset{\boldsymbol{u}}{\operatorname{argmax}}\Big(\frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{u}^\top(\boldsymbol{x}_i - \boldsymbol{m})]^2\Big) = \underset{\boldsymbol{u}}{\operatorname{argmax}}(\boldsymbol{u}^\top\boldsymbol{\Sigma}\boldsymbol{u}) \quad \text{with} \quad ||\boldsymbol{u}|| = 1,$$

$$\text{where} \quad \boldsymbol{m} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i, \quad \boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{m})^\top$$

$\square$

b) We use the Lagrange Multiplier method from class for ease of calculation with:

$$f(u_1,\dots,u_n)=f(\boldsymbol{u})=\boldsymbol{u}^\top\boldsymbol{\Sigma}\boldsymbol{u}=\begin{bmatrix}u_1 & \dots & u_n\end{bmatrix}\begin{bmatrix}s_{11} & \dots & s_{1n}\\ \vdots & \ddots & \vdots\\ s_{n1} & \dots & s_{nn}\end{bmatrix}\begin{bmatrix}u_1\\ \vdots\\ u_n\end{bmatrix}$$

$$g(u_1,\dots,u_n)=g(\boldsymbol{u})=1-||\boldsymbol{u}||^2=1-\boldsymbol{u}\cdot\boldsymbol{u}=1-\sum_{i=1}^n u_i^2$$

where $\boldsymbol{\Sigma}=(s_{ij})_{n\times n}$, and the Lagrangian is $\mathcal{L}(\boldsymbol{u},\lambda):=f(\boldsymbol{u})+\lambda g(\boldsymbol{u})$. Functions f,g are smooth and the constraint $||\boldsymbol{u}||=1$ in the proposed question is equivalent to $g(\boldsymbol{u})=1-||\boldsymbol{u}||=0$. Then we apply the method:

$$0=\nabla_\lambda\mathcal{L}(\boldsymbol{u},\lambda)=\frac{\partial}{\partial\lambda}\mathcal{L}(\boldsymbol{u},\lambda)=g(\boldsymbol{u})=1-||\boldsymbol{u}||^2$$

which is, of course, true by the definition of the constraint function $g(\boldsymbol{u})$. We in fact need the equation coming from $0=\nabla_{\boldsymbol{u}}\mathcal{L}$, so we calculate:

$$\begin{aligned}0&=\nabla_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{u},\lambda)\\&=\nabla_{\boldsymbol{u}}f(\boldsymbol{u})+\lambda\cdot\nabla_{\boldsymbol{u}}g(\boldsymbol{u})\end{aligned}$$

First, we can quickly compute $\nabla_{\boldsymbol{u}}g(\boldsymbol{u})=\nabla_{\boldsymbol{u}}(1-||\boldsymbol{u}||^2)$ since we already have the coordinate-wise expression $1-\sum u_i^2$ for the function g:

$$\begin{aligned}\nabla_{\boldsymbol{u}}g(\boldsymbol{u})&=\nabla_u(1-||\boldsymbol{u}||^2)\\&=\nabla_u\Big(1-\sum_{i=1}^n u_i^2\Big)\\&=\Big[\tfrac{\partial}{\partial u_1}\Big(1-\textstyle\sum_{i=1}^n u_i^2\Big)\quad\dots\quad\tfrac{\partial}{\partial u_k}\Big(1-\textstyle\sum_{i=1}^n u_i^2\Big)\quad\dots\quad\tfrac{\partial}{\partial u_n}\Big(1-\textstyle\sum_{i=1}^n u_i^2\Big)\Big]^\top\\&=\Big[-\textstyle\sum_{i=1}^n\tfrac{\partial}{\partial u_1}(u_i^2)\quad\dots\quad-\textstyle\sum_{i=1}^n\tfrac{\partial}{\partial u_k}(u_i^2)\quad\dots\quad-\textstyle\sum_{i=1}^n\tfrac{\partial}{\partial u_n}(u_i^2)\Big]^\top\\&=-\Big[\textstyle\sum_{i=1}^n 2u_i\tfrac{\partial u_i}{\partial u_1}\quad\dots\quad\textstyle\sum_{i=1}^n 2u_i\tfrac{\partial u_i}{\partial u_k}\quad\dots\quad\textstyle\sum_{i=1}^n 2u_i\tfrac{\partial u_i}{\partial u_n}\Big]^\top\\&=-\big[\textstyle\sum_{i=1}^n 2u_i\delta_{i1}\quad\dots\quad\textstyle\sum_{i=1}^n 2u_i\delta_{ik}\quad\dots\quad\textstyle\sum_{i=1}^n 2u_i\delta_{in}\big]^\top\\&=-\begin{bmatrix}2u_1 & \dots & 2u_k & \dots & 2u_n\end{bmatrix}^\top\\&=(-2)\cdot\begin{bmatrix}u_1\\ \vdots\\ u_n\end{bmatrix}\\&=-2\boldsymbol{u}\end{aligned}$$

Secondly, we will calculate the gradient of f. Observe the following equations before we continue:

$$\begin{aligned}
\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u} &= \begin{bmatrix} u_1 & \dots & u_n \end{bmatrix} \begin{bmatrix} s_{11} & \dots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{n1} & \dots & s_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^n u_i s_{i1} & \dots & \sum_{i=1}^n u_i s_{in} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \\
&= \sum_{j=1}^n \sum_{i=1}^n (u_i s_{ij} u_j) \\
&= \sum_{j=1}^n \sum_{i=1}^n s_{ij} (u_i u_j)
\end{aligned}$$

Now we can directly compute the gradient of f with respect to $\mathbf{u}$:

$$\begin{aligned}
\nabla_{\boldsymbol{u}} f(\boldsymbol{u}) &= \nabla_{\boldsymbol{u}} (\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}) \\
&= \left[\tfrac{\partial}{\partial u_1} \sum_{j=1}^n \sum_{i=1}^n s_{ij}(u_i u_j) \quad \dots \quad \tfrac{\partial}{\partial u_n} \sum_{j=1}^n \sum_{i=1}^n s_{ij}(u_i u_j) \right]^\top \\
&= \left[\tfrac{\partial}{\partial u_k} \sum_{j=1}^n \sum_{i=1}^n s_{ij}(u_i u_j) \right]_{1 \le k \le n}^\top \\
&= \left[\sum_{j=1}^n \sum_{i=1}^n s_{ij} \cdot \tfrac{\partial}{\partial u_k}(u_i u_j) \right]_{1 \le k \le n}^\top \\
&= \left[\sum_{j=1}^n \sum_{i=1}^n s_{ij} \cdot \tfrac{\partial u_i}{\partial u_k} \cdot u_j + \sum_{j=1}^n \sum_{i=1}^n s_{ij} \cdot u_i \cdot \tfrac{\partial u_j}{\partial u_k} \right]_{1 \le k \le n}^\top \\
&= \left[\sum_{j=1}^n \sum_{i=1}^n s_{ij} \cdot \tfrac{\partial u_i}{\partial u_k} \cdot u_j \right]_{1 \le k \le n}^\top + \left[\sum_{j=1}^n \sum_{i=1}^n s_{ij} \cdot u_i \cdot \tfrac{\partial u_j}{\partial u_k} \right]_{1 \le k \le n}^\top \\
&= \left[\sum_{j=1}^n \sum_{i=1}^n s_{ij} \cdot \delta_{ik} \cdot u_j \right]_{1 \le k \le n}^\top + \left[\boldsymbol{\sum_{i=1}^n \sum_{j=1}^n} u_i \cdot s_{ij} \cdot \delta_{jk} \right]_{1 \le k \le n}^\top \\
&= \left[\sum_{j=1}^n s_{kj} \cdot u_j \right]_{1 \le k \le n}^\top + \left[\sum_{i=1}^n u_i \cdot s_{ik} \right]_{1 \le k \le n}^\top \\
&= \begin{bmatrix} \sum_{j=1}^m s_{1j} u_1 \\ \vdots \\ \sum_{j=1}^m s_{nj} u_n \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^m u_i s_{i1} & \dots & \sum_{i=1}^n u_i s_{in} \end{bmatrix}^\top \\
&= \boldsymbol{\Sigma u} + (\boldsymbol{u}^\top \boldsymbol{\Sigma})^\top \\
&= \boldsymbol{\Sigma u} + \boldsymbol{\Sigma}^\top \boldsymbol{u} \\
&= \boldsymbol{\Sigma u} + \boldsymbol{\Sigma u} \quad \text{(since } \boldsymbol{\Sigma} \text{ is symmetric)} \\
&= 2\boldsymbol{\Sigma u}
\end{aligned}$$

Finally, we can insert our findings into the original equation:

$$
\begin{aligned}
0 &= \nabla_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{u}, \lambda) \\
0 &= \nabla_{\boldsymbol{u}} f(\boldsymbol{u}) + \lambda \cdot \nabla_{\boldsymbol{u}} g(\boldsymbol{u}) \\
0 &= \nabla_{\boldsymbol{u}} (\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}) + \lambda \cdot \nabla_{\boldsymbol{u}} (1 - ||\boldsymbol{u}||^2) \\
0 &= 2\boldsymbol{\Sigma}\boldsymbol{u} - \lambda \cdot 2\boldsymbol{u}
\end{aligned}
$$

Hence, we finally got

$$\boldsymbol{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}$$

that shows that the Langrage multiplier λ in our maximization problem is actually one of the eigenvalues of the covariance matrix $\boldsymbol{\Sigma}$, and the vector $\boldsymbol{u}$ that maximizes the original expression f is in fact an eigenvector with $||\boldsymbol{u}|| = 1$ by the original constraint.

Now we ask: Which one of the eigenvalues is actually equal to the Lagrange multiplier λ? For all the eigenvalues, we have the following equation:

$$f(\boldsymbol{u}) = \boldsymbol{u}^\top \Sigma \boldsymbol{u} = \boldsymbol{u}^\top (\lambda \boldsymbol{u}) = \lambda(\boldsymbol{u}^\top \boldsymbol{u}) = \lambda$$

After we realize that this last equation is true for any eigenvalue λ and its eigenvectors $\boldsymbol{u}$, for our specific eigenvalue λ which also happens to be the Lagrange multiplier, the corresponding vectors $\boldsymbol{u}$ are also the maximizing parameters of f as the Lagrangian method suggests. Therefore, our λ must be the biggest eigenvalue among the other real eigenvalues of the covariance matrix $\boldsymbol{\Sigma}$ because, for any other pair λ' and $\boldsymbol{u}'$, we have

$$\lambda' = f(\boldsymbol{u}') \leq f(\boldsymbol{u}) = \lambda.$$

So we proved that finding the maximizing value for $\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}$ with $||\boldsymbol{u}|| = 1$ can be reformulated as finding the eigenvectors $\boldsymbol{u}$ associated with the highest eigenvalue λ of the covariance matrix $\boldsymbol{\Sigma}$.

$\square$

Exercise 2

a) We use the definition of trace and $\mathbf{\Sigma}$ to show the upper bound. We have the following great equation:

$$\sum_{i=1}^{d} \mathbf{\Sigma}_{ii} =: \operatorname{tr}(\mathbf{\Sigma}) = \sum_{j=1}^{d} \lambda_j \geq \lambda_1$$

Here $\mathbf{\Sigma}$ is positive semi-definite, so we have $\lambda_j \geq 0$ for all eigenvalues λ_i's. Thus, the last inequality is also true.

$\square$

b) We define k to be the index of maximum on the diagonal and use the objective function of PCA to show the lower bound. Using the above formulas for $\boldsymbol{u}^\top \mathbf{\Sigma} \boldsymbol{u}$, we have:

$$\begin{aligned}
\max_{i=1}^{d} \mathbf{\Sigma}_{ii} &= \mathbf{\Sigma}_{kk} \\
&= \boldsymbol{e}_k^\top \mathbf{\Sigma} \boldsymbol{e}_k \\
&\leq \underset{\boldsymbol{u}}{\operatorname{argmax}}(\boldsymbol{u}^\top \mathbf{\Sigma} \boldsymbol{u}) \qquad (||\boldsymbol{e}_k|| = ||\boldsymbol{u}|| = 1) \\
&= \lambda_1 \qquad \text{(By the objective function of PCA)}
\end{aligned}$$

which uses Exercise 1.b for the last equation as λ_1 is the greatest eigenvalue.

$\square$

Exercise 3

(a)

$$\begin{aligned}
\mathcal{E}_k(w^{(t+1)}) &= \frac{|w^{(t+1)^\top} u_k|}{|w^{(t+1)^\top} u_1|} \\
&= \frac{|(\mathbf{\Sigma} w^{(t)}/||\mathbf{\Sigma} w^{(t)}||)^\top u_k|}{|(\mathbf{\Sigma} w^{(t)}/||\mathbf{\Sigma} w^{(t)}||)^\top u_1|} \\
&= \frac{|w^{(t)^\top} \mathbf{\Sigma}^\top / ||\mathbf{\Sigma} w^{(t)}|| u_k|}{|w^{(t)^\top} \mathbf{\Sigma}^\top / ||\mathbf{\Sigma} w^{(t)}|| u_1|} \\
&= \frac{|w^{(t)^\top} \mathbf{\Sigma}^\top u_k|}{|w^{(t)^\top} \mathbf{\Sigma}^\top u_1|} \\
&= \frac{|w^{(t)^\top} \mathbf{\Sigma} u_k|}{|w^{(t)^\top} \mathbf{\Sigma} u_1|} \text{ (Since } \mathbf{\Sigma} \text{ is symmetric)}
\end{aligned}$$

$\square$

(b)

$$\begin{aligned}\mathcal{E}_k(w^{(t+1)}) &= \frac{|{w^{(t)}}^\top \mathbf{\Sigma} u_k|}{|{w^{(t)}}^\top \mathbf{\Sigma} u_1|} \\ &= \frac{|{w^{(t)}}^\top \lambda_k u_k|}{|{w^{(t)}}^\top \lambda_1 u_1|} \\ &= \frac{|\lambda_k|}{|\lambda_1|}\frac{|{w^{(t)}}^\top u_k|}{|{w^{(t)}}^\top u_1|} \\ &= \frac{|\lambda_k|}{|\lambda_1|}\mathcal{E}_k(w^{(t)})\end{aligned}$$

$\square$

(c) Proving $\mathcal{E}_k(w^{(t+1)}) = (\frac{|\lambda_k|}{|\lambda_1|})^i \mathcal{E}_k(w^{(t+1-i)})$ for $1 \le i \le t+1$ by Induction:

Base Case $i = 1$**:** Shown in **(b)**
Induction Hypothesis for $i < t+1$**:**

$$\begin{aligned}\mathcal{E}_k(w^{(t+1)}) &= (\frac{|\lambda_k|}{|\lambda_1|})^t \mathcal{E}_k(w^{(t+1-t)}) \\ &= (\frac{|\lambda_k|}{|\lambda_1|})^t \frac{|\lambda_k|}{|\lambda_1|}\mathcal{E}_k(w^{(0)}) \\ &= (\frac{|\lambda_k|}{|\lambda_1|})^{t+1} \mathcal{E}_k(w^{(0)})\end{aligned}$$

Here the first equality comes from the Induction Hypothesis with $i = t$, and the second equation is inserted from the above Exercise 3.b with the value $t + 1 = 1$.
$\square$

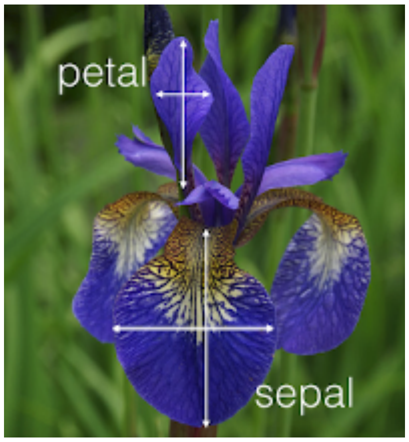
Exercises for the course
Machine Learning for Data Science
Winter Semester 2024/25

G. Montavon
Institute of Computer Science
Department of Mathematics and Computer Science
Freie Universität Berlin

Exercise Sheet 3 (programming part)

In [1]:
```
import numpy
import sklearn
import sklearn.datasets
import matplotlib
%matplotlib inline
from matplotlib import pyplot as plt
```

In this exercise we perform a principal component analysis of the Iris dataset, a famous dataset from Fisher (1936) which one can access from sklearn. Each instance of the dataset is an iris plant which comes with four measurements (1. sepal length in cm, 2. sepal width in cm, 3. petal length in cm, 4. petal width in cm). What these measurements correspond to is depicted in the image below.



In addition to these measurements, the dataset also includes for each instance the type of iris plant (iris setosa, iris versicolour, iris virginica) which we treat here as metadata. Overall, the Iris dataset has 150 instances, and can be stored as an array of size 150 x 4.

The following cell loads the dataset. Additionally, some logarithmic transformation of the input features is performed so that a difference between e.g. 5mm and 6mm is given roughly the same importance as a difference between 5cm and 6cm. Such log-scaling gives more focus on features measuring smaller objects, compared to an approach based on the raw measurements. Also, we add a small increment of 1mm before applying the log-transform to ignore very small quantities that cannot be precisely measured.

```
In [2]: dataset = sklearn.datasets.load_iris()

        X = numpy.log(0.1+dataset['data'])
        T = dataset['target']

        target_names = dataset['target_names']
        feature_names = dataset['feature_names']

        X = X - X.mean(axis=0)

        N,d = X.shape
```

Note that one must keep in mind that our features have been log-transformed in any of the subsequent analyses and interpretations. A first basic analysis that can be performed is to measure how much dispersion there is in the data. The total variance of the data can be computed by the code below:

```
In [3]: stot = (X**2).mean(axis=0).sum()
        print('Total variance: %.3f'%stot)
```

```
Total variance: 1.027
```

Exercise 3 (10 + 10 + 10 P)

We now would like to shed more light into the data by performing a principal component analysis (PCA). The exercise will consist of implementing PCA and then perform various analyses based on the learned model.

Implementing PCA

The simplest way of implementing PCA, is through an eigendecomposition of the data covariance matrix Σ, followed by solving the eigenvalue equation $\Sigma \boldsymbol{u} = \lambda \boldsymbol{u}$.

Task: Implement the PCA algorithm. Your code should return a tuple consisting of (1) a vector `L` containing the d eigenvalues, in decreasing order, and (2) a matrix `U` where each column contains the eigenvector associated to the eigenvalue.

Eigenvectors in this matrix should be ordered according to the eigenvalues, i.e. in a way that, for all indices $i = 1 \dots d$, the column `U[:,i]` contains the eigenvector associated to the eigenvalue `L[i]`.

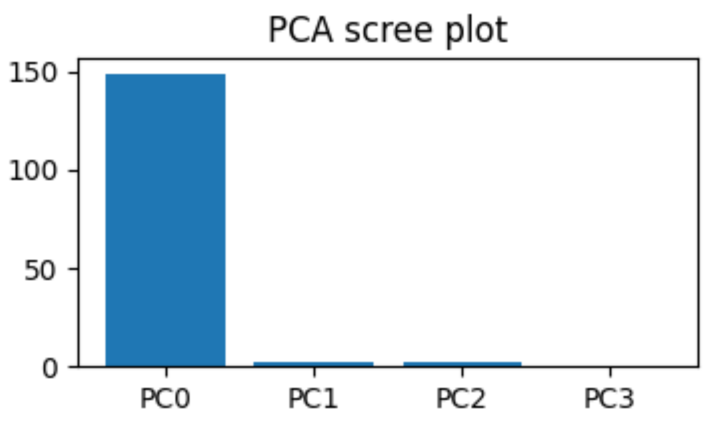
```
In [4]: # ------------------------------------------
        def PCA(M):
            eigval, eigvec = numpy.linalg.eigh(M.transpose() @ M)
            eigval = eigval[::-1]
            eigvec[:,0], eigvec[:,3] = eigvec[:,3], eigvec[:,0]
            eigvec[:,1], eigvec[:,2] = eigvec[:,2], eigvec[:,1]
```

```
    return eigval, eigvec
L, U = PCA(X)
# ------------------------------------------
# import solution
# L,U = solution.PCA(X)
# ------------------------------------------
```

As mentioned in the lecture, PCA can be seen as a way of decomposing the total variance in the data. This decomposition of variance offered by PCA can be rendered in a PCA scree plot. (The following code further verifies that the PCA scree plot corresponds to a decomposition of the total variance, i.e. that the sum of plotted eigenvalues should be equivalent to the measured total variance.)

In [5]:
```
plt.figure(figsize=(4,2))
plt.title ('PCA scree plot')
plt.bar(numpy.arange(d),L)
plt.xticks(range(d),labels=['PC%d'%i for i in range(d)])
plt.show()

print('Total variance: %.3f'%stot)
print('Sum of eigenvalues: %.3f'%L.sum())
```



```
Total variance: 1.027
Sum of eigenvalues: 153.984
```

We observe that the first principal component captures most of the variance in the data. This suggests that one or a few strongly covariate features capture most of the variations in the data.

Understanding the First Principal Component

We now would like to gain understanding of this highly explanatory first principal component, specifically, to which input features this component responds. For this, we recall that the projection of data on a principal component has the dot-product form

$$z = \boldsymbol{u}^\top \boldsymbol{x} = u_1 x_1 + u_2 x_2 + \cdots + u_d x_d$$

where in our case, $\boldsymbol{u}$ is the principal component stored in our column vector `U[:,0]` .

Task: Write code that displays this projection operation instantiated to our particular problem formulation and PCA model.

In [6]:
```
# ------------------------------------------
proj = "z = "
for i in range(len(U[:,0])):
    proj += f"({U[:,0][i]:.3f}) * log(0.1 + {feature_names[i]})\n+ "
print(proj[:-3])
# ------------------------------------------
# import solution
# solution.printformula(U,feature_names)
# ------------------------------------------
```

```
z = (-0.115) * log(0.1 + sepal length (cm))
+ (0.064) * log(0.1 + sepal width (cm))
+ (-0.561) * log(0.1 + petal length (cm))
+ (-0.817) * log(0.1 + petal width (cm))
```

As can be observed from the formula above, the first principal component responds mostly to a combination of petal length and petal width. It is on the other hand rather insensitive to sepal-related features.

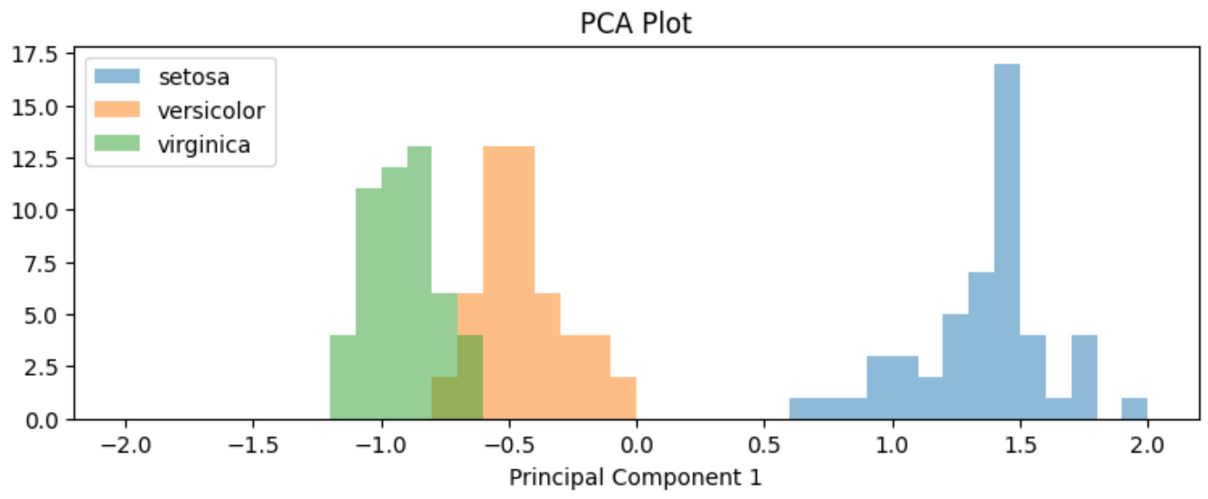
Understanding the First Principal Component

We now would like to get understanding of the data projected on this principal component. The main advantage of looking at the PCA space instead of the feature space is that its dimensionality has been reduced, thereby allowing for using tools such as simple histograms without performing a priori feature selection. The following code projects our data on the first principal component:

In [7]:
```
Z = X.dot(U[:,0])
```

Task: Write code that visualizes the projected data in the form of a histogram. Specifically, your visualization should split data according to the meta-data (here, the distinct plant species), and produce for each of them a distinct histogram with a specific color.

In [8]:
```
# ------------------------------------------
data = {str(x): [] for x in target_names}
for i in range(len(T)): data[target_names[T[i]]].append(Z[i])
plt.figure(figsize=(9, 3))
plt.title("PCA Plot")
for key in data:
    plt.hist(data[key], bins=numpy.arange(-2.0, 2.1, 0.1), label=key, alpha=
plt.xlabel("Principal Component 1")
plt.legend()
plt.show()
# ------------------------------------------
# import solution
# solution.PCAplot(Z,T,target_names)
# ------------------------------------------
```



From this analysis, we can observe that the principal component represent meaningful variations in the data, in particular, species appear to be separable. The Iris setosa is separated from the other two species by a large margin, whereas the two other species are also separated to a certain extent, but not fully.