Home Assignment 5
Machine Learning A

FEDERICO FIORIO

October 10, 2022

1 Principal Component Analysis (50 points)

1.1 PCA and centering (15 points)

Given what is already proven by the book, where Z is the centered matrix:

By direct calculation, one can verify that $\mathrm{Z} = \mathrm{X} - \mathbf{1}\bar{\mathbf{x}}^{\mathrm{T}}$. Hence,

$$\bar{\mathbf{z}} = \frac{1}{N}\mathrm{Z}^{\mathrm{T}}\mathbf{1} = \frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathbf{1} - \frac{1}{N}\bar{\mathbf{x}}\mathbf{1}^{\mathrm{T}}\mathbf{1} = \bar{\mathbf{x}} - \frac{1}{N}\bar{\mathbf{x}} \cdot N = \mathbf{0},$$

Figure 1: Book's proof Logistic regression chapter from Abu-Mostafa et al., Learning from Data (amlbook.com)

From this proof, we can see that the mean vector $\tilde{\mathbf{z}}$ is equal to 0.
We can also see that we obtain that as:
$\frac{1}{N}Z^T\mathbf{1}$
That operation translates as a sum over the rows of the matrix Z divided by N.
This operation it's basically a linear combination of the rows of Z which is the matrix on which we want to prove that the rank is at most d-1.
Given what was just said here, we can say that if we sum all the rows of the matrix Z and divide by N we get a vector of zeros, which means that at least 1 row in Z is linearly dependent from the others, which means that the rank of Z is at most d-1 because there could be at most d-1 linearly indipendent rows.

1.2 Explained variance and Bessel's correction (5 points)

To obtain the eigenvectors and eigenvalues used in PCA, we firstly compute the covariance matrix which as we can see is divided by $\frac{1}{N}$

$$\boldsymbol{S} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^{\mathsf{T}} \quad .$$

Figure 2: covariance matrix from slides

Given the property of eigenvalues:
$cMv = c\lambda v$ where c is a scalar.
In the above formula, c = $\frac{1}{N}$ but we can easily change it to $\frac{1}{N-1}$ and the eigenvalue will also change by that constant c, but given the explained variance, we can easily see that it won't change even if the constant c changes:
$\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{d}\lambda_i} = \frac{\sum_{i=1}^{k}c\lambda_i}{\sum_{i=1}^{d}c\lambda_i} = \frac{c\sum_{i=1}^{k}\lambda_i}{c\sum_{i=1}^{d}\lambda_i} = \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$

1.3 PCA in practice

1.3.1 Explained variance (15 points)

Do the PCA:

```
pca = PCA()
pca.fit(X)
eigenvalues = pca.singular_values_**2
explained_var_comp = pca.explained_variance_ratio_
```

```
plt.yscale('log')
plt.plot(eigenvalues / np.sum(eigenvalues))
```

```
[<matplotlib.lines.Line2D at 0x2428b0e9690>]
```

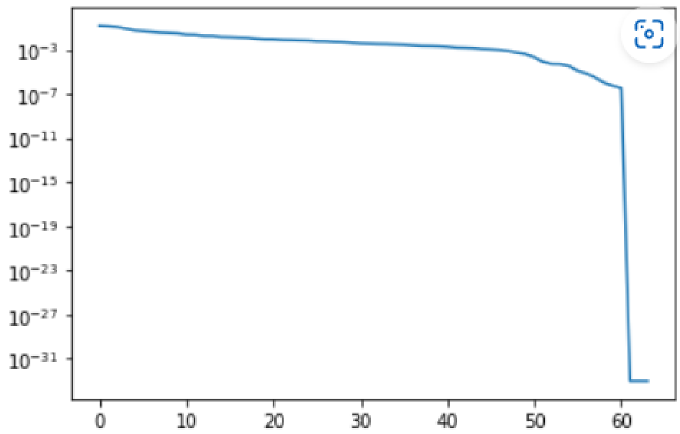

Figure 3: Plot of eigenvalues / sum of eigenvalues

As we can see from the following image, 10 principal components are not enough to explain 80% of variance

Find out if 10 components are enough to explain 80% of the variance:

```
np.sum(explained_var_comp[:10]) #not enough
```

```
0.7382267688459533
```

```
plt.plot(np.cumsum(explained_var_comp[:10]))
```

```
[<matplotlib.lines.Line2D at 0x242ffd875e0>]
```

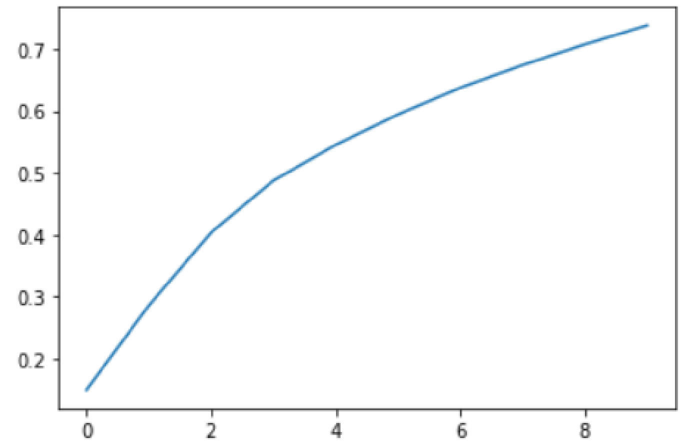

Figure 4: Explained variance of 10 principal components

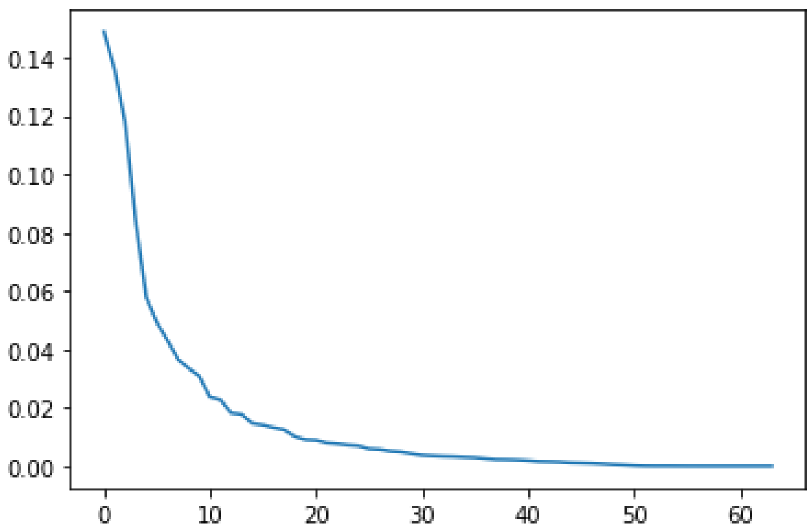

Figure 5: Eigenspectrum, not sure if it's needed

1.3.2 Eigendigits (15 points)

```
for i in range(5):
    plt.imshow(pca.components_[i].reshape(imshape))
    plt.show()
```

Figure 6: Code for plotting

Plots:

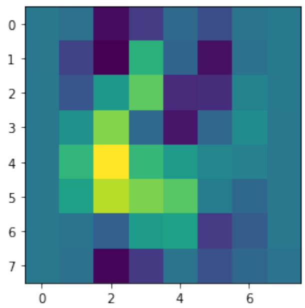

Figure 7: Eigendigit 1

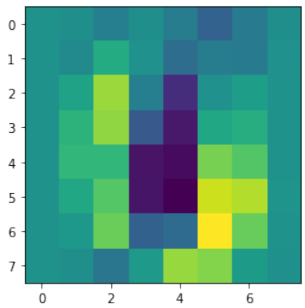

Figure 8: Eigendigit 2

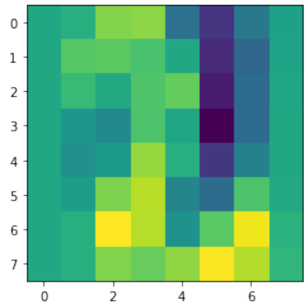

Figure 9: Eigendigit 3

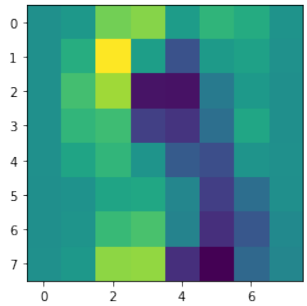

Figure 10: Eigendigit 4

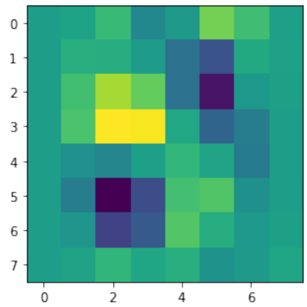

Figure 11: Eigendigit 5

2 Logistic Regression in PyTorch (50 points)

2.1 PyTorch self-study (0 points)

2.2 Logistic Regression in PyTorch (50 points)

I find it more understandable if I attach directly the screenshots of the code instead of just the lines that I should have written.

```
class LogisticRegressionPytorch(nn.Module):
    def __init__(self, d, m):
        super(LogisticRegressionPytorch, self).__init__()
        # SOMETHING MISSING HERE
        self.linear = nn.Linear(d, m)

    def forward(self, x):
        # SOMETHING MISSING HERE
        outputs = self.linear(x)
        return outputs

logreg_pytorch = LogisticRegressionPytorch(d, m)
print(logreg_pytorch)
```

```
LogisticRegressionPytorch(
  (linear): Linear(in_features=2, out_features=4, bias=True)
)
```

Figure 12: Model definition, I did not apply sigmoid layer because crossEntropy already apply that

Model training and evaluation

First we have to define a loss function. *Double check that output of the network and the expected input of the loss function match!*

```
optimizer = optim.Adam(logreg_pytorch.parameters(), lr=0.01)
# DEFINITION OF LOSS FUNCTION MISSING
CEL = nn.CrossEntropyLoss()
```

Figure 13: Loss function

```
no_epochs = 10000  # Number of training steps
X_train_T = torch.Tensor(X_train)  # Automatically casts to foat
y_train_T = torch.from_numpy(y_train)  # Does not cast to float
#y_train_T = y_train_T.long()
for epoch in range(no_epochs):  # Loop over the dataset multiple times
    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward + backward + optimize
    outputs = logreg_pytorch(X_train_T)
    # SOMETHING MISSING HERE
    loss = CEL(outputs,  y_train_T.long()) #probabilities, y_train, if they are long the softmax is done autom.
    # SOMETHING MISSING HERE
    loss.backward()

    optimizer.step()
```

Figure 14: Training loop

```
ws_torch = logreg_pytorch.linear.weight.detach().numpy()
# SOMETHING MISSING HERE
bs_torch = logreg_pytorch.linear.bias.detach().numpy()
```

Figure 15: Getting parameters of the model

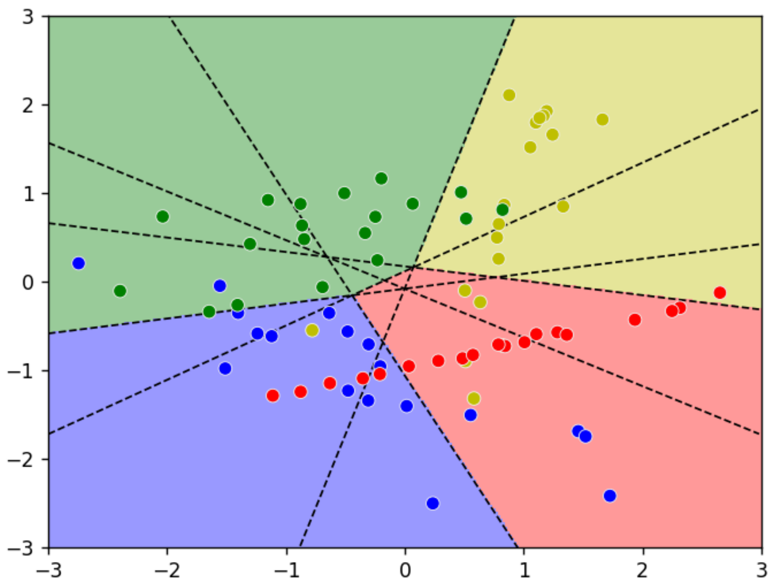

Figure 16: visualizing the predictions, it is the same as the sklearn implementation (I know the points are random but it's just to give an idea)