Exercise 4.0 — Principal Component Analysis, k-Means and Support Vector Machines

Artificial Intelligence For Robotics

Abel Gawel

Spring Semester 2017

1 Introduction

Imagine you are working with autonomous Aerial Vehicles. Localization is a crucial component for your robot to perform it's tasks. Furthermore, the Aerial vehicles are tasked with different missions depending on the environment they are exploring, i.e., perform inspection tasks in industrial environment versus surveillance in warehouse environments. For this we want to enable the robots to distinguish which of those environments they are presently exploring. Therefore you will implement a place classification algorithm based on Principal Component Analysis (PCA), K-means clustering, and Support Vector Machines (SVMs) applied to the camera data of an Aerial robot.

2 Principal Component Analysis (PCA)

In this exercise you will implement a dimensionality reduction algorithm based on the Principal Components of image data descriptors to extract feature vectors for place categorization. The repetitions in this exercise sheet are based on [1].

Remember that many data can be distinguished using much lower dimensionality than the original data. The PCA algorithm aims to achieve a dimensionality reduction by finding a subspace that maximizes the variance between the dimensions of this subspace and projecting the original data onto this subspace.

Consider a dataset of datapoints $\mathbf{x}_n$ with $n = 1, ..., N$ and $\mathbf{x}_n \in \mathbb{R}^D$ that we aim to project to a lower dimensionality $M < D$. We can define this problem for a single dimension as the maximization of the variance between the projected datapoints $\mathbf{u}^\top \mathbf{x}_n$ with $\mathbf{u}$ being the D-dimensional projection vector.

$$\mathbf{u} = \arg\max_{\mathbf{u}} \mathbf{u}^\top \boldsymbol{\Omega} \mathbf{u} \tag{1}$$

with covariance $\boldsymbol{\Omega}$ and variance $\mathbf{u}^\top \boldsymbol{\Omega} \mathbf{u}$. However, to constraint the maximization we introduce our constraint to have $\mathbf{u}$ as unit vector, i.e., $\mathbf{u}^\top \mathbf{u} = 1$ and introduce this constraint as a Lagrange multiplier λ. We aim to maximize the expression:

$$\mathbf{u}^\top \boldsymbol{\Omega} \mathbf{u} + \lambda(1 - \mathbf{u}^\top \mathbf{u}) \tag{2}$$

By setting the derivative with respect to $\mathbf{u}$ to zero, we yield

$$\boldsymbol{\Omega} \mathbf{u} = \lambda \mathbf{u} \tag{3}$$

Reordering this equation, we see that the variance at this point is given by

$$\mathbf{u}^\top \boldsymbol{\Omega} \mathbf{u} = \lambda, \tag{4}$$

where λ is the largest eigenvalue and $\boldsymbol{u}$ the corresponding eigenvector. We can then apply this iteratively with the eigenvectors corresponding to the eigenvalues in descending order to yield the the principal components.

2.1 Assignments

For this first assignment, we aim to extract the principal components from a collection of image features. The reasoning here is that we aim to find a very simple representation for the environment, i.e., a vector with few dimensions per image. Therefore, we first extract a large amount of image descriptors, i.e., ORB descriptors, from the images, see Fig. 1. However, for a simple place categorization we do not require the full feature vector, i.e., $n_{feat} = n_{kp} n_{ORB}$, with the number of keypoints per image n_{kp} and the number of feature dimension n_{ORB}. Here, our input feature vector has a dimensionality of $n_{feat} = 100 \times 32 = 3200$. As the evaluation of all dimensions in a brute force matching would be a computationally expensive operation, we wish to drastically reduce the dimensionality of the feature vector to the first 5 Principal Components. The images are from the popular EuRoC dataset [2].

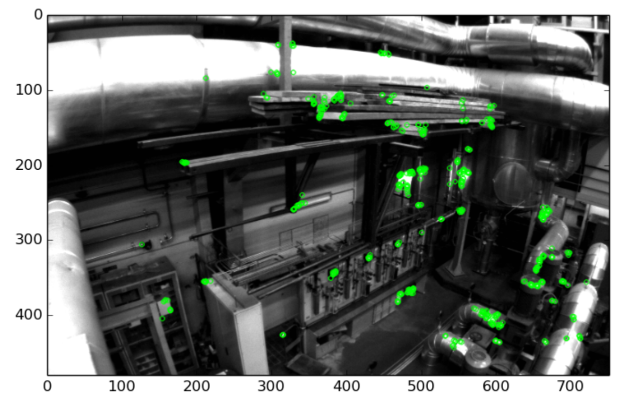

Figure 1: Image of the EuRoC dataset [2] with detected ORB keypoints overlay.

Tasks: Follow the steps of the repetition to complete the sections indicated with `#TODO` in `pca_main.py`.

1. Compute the covariance of the input data.
2. Compute eigenvalues and eigenvectors of the covariance matrix.
3. Extract the first 5 principal components and normalize the features afterwards.
4. Store your result in the $n \times 5$ dimensional matrix `pca_features` and save it to the file `results_pca.txt`.

The program will plot the first 2 principal components for you. You should expect a result similar to Fig. 2.

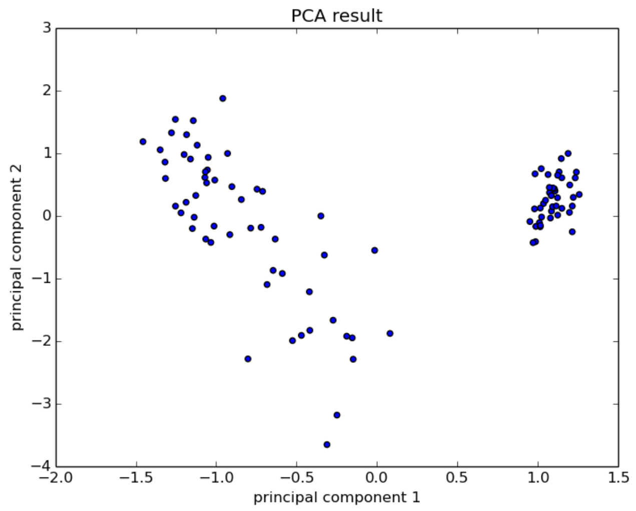

Figure 2: Expected result of PCA on ORB features.

3 K-Means clustering

Another important aspect in feature extraction is to build groups of datapoints that lie close in feature space. For instance, after reducing the data to a lower dimensional feature space as done in the previous exercise, our next step is to form clusters of samples in this feature space. The problem is to partition the datapoints x_n with $n = 1, ..., N$ into K clusters, where K is given a priori, such that the sum of squared distances of each point to its assigned cluster center $\boldsymbol{\mu}_k$ is minimal, yielding the objective function J, i.e.,

$$J = \arg\min_{r_{nk}, \boldsymbol{\mu}_k} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \tag{5}$$

with r_{nk} being the assignment of datapoint $\mathbf{x}_n$ to cluster k, i.e., $r_{nk} = 1$ and $r_{nj} = 0$ for $j \neq k$. Remember from the lecture that this optimization problem is NP-hard. However, we can treat this minimization problem as an optimization problem with two successive steps optimizing r_{nk} and $\boldsymbol{\mu}_k$ using Lloyd's algorithm. First, we choose an initial value for $\boldsymbol{\mu}_k$ and optimize J with respect to r_{nk}, while keeping $\boldsymbol{\mu}_k$ fixed. We then optimize J with respect to $\boldsymbol{\mu}_k$ while keeping r_{nk} fixed. This process is repeated until convergence.

Since the optimization problem is linear in r_{nk}, our first optimization task is independent of n and we can solve for each r_{nk} separately, i.e.,

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The second step of the optimization can be determined by holding r_{nk} fixed and setting the derivative of J with respect to $\boldsymbol{\mu}_k$ to zero, yielding

$$2\sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0 \tag{7}$$

which we can solve for $\boldsymbol{\mu}_k$

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}}. \tag{8}$$

3.1 Assignments

For this assignment, we aim to partition the datapoints represented as 5-dimensional features into clusters.

Tasks: Follow the steps of the repetition to complete the sections indicated with `#TODO` in `k_means_main.py`.

Note: For this exercise you should load the provided feature data and not re-use the computed features of the first exercise!

1. Complete the clustering function which assigns datapoints x_n to clusters u_k using equation 6.
2. Complete the recalculation of the centers μ using equation 8.
3. Cluster the datapoints into 3 clusters. When running the clustering multiple times, you may realize that the cluster centers do not always converge to a good solution. Perform a different initialization of your algorithm to prevent convergence to local minima, i.e., faulty clusters as discussed in the lecture, e.g., using k-means++ or evaluating multiple initializations. **Do not manually tweak the starting point, but**

use an algorithmic solution. We will test your algorithm on different evaluation data!

4. Save your 3 cluster centers $\boldsymbol{\mu}_k$ in the 3×5 dimensional matrix `mu` and save it to the file `results_k_means.txt`.

The program will plot the clustering on the first 2 principal components for you. You should expect a result similar to Fig. 3.

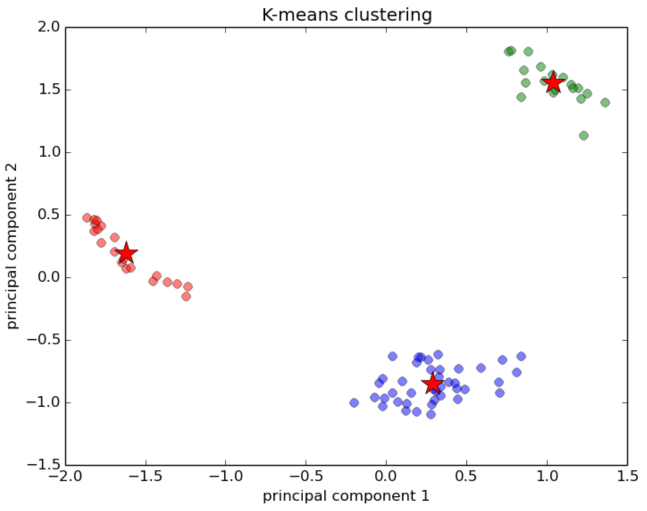

Figure 3: Expected result of k-means.

4 Support Vector Machines (SVMs)

After partitioning our data into classes, we now want to build a classification system for evaluating new datapoints and sorting them in one or the other class. A popular algorithm for (non-) linear classification problems on a multi-dimensional feature-space is the Support Vector Machines (SVMs) algorithm. It estimates a hyperplane to separate the datapoints. Moreover, it is possible to produce non-linear decision boundaries by using the kernel-trick. To begin with a linear SVM, the two-class classification task is formulated as follows

$$u = \mathbf{w}^\top \mathbf{x} + \mathbf{b}, \tag{9}$$

with the normal vector of the hyperplane $\mathbf{w}$, the input vector $\mathbf{x}$, and the intercept $\mathbf{b}$. The separating hyperplane will go through $u = 0$ and the nearest points of the two classes will lie on plane at $u = \pm 1$.

Remember the optimization problem in dual form for estimating the separating hyperplane, as introduced in the lecture, where the objective function Ψ only depends on a set of Lagrange multipliers $\boldsymbol{\alpha}_i \geq 0$.

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\Psi}(\boldsymbol{\alpha}) = \min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j (\mathbf{x}_i \mathbf{x}_j) \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j - \sum_{i=1}^{N} \boldsymbol{\alpha}_i, \tag{10}$$

where N is the number of training examples, and $y_i = \pm 1$ the classification results. $\boldsymbol{w}$ and $\boldsymbol{b}$ can be directly estimated from the Lagrange multipliers

$$\mathbf{w} = \sum_{i=1}^{N} y_i \boldsymbol{\alpha}_i \mathbf{x}_i,\ \boldsymbol{b} = \boldsymbol{w}\mathbf{x}_k - y_k \text{ for some } \boldsymbol{\alpha}_k > 0 \tag{11}$$

The SVM can be further generalized to non-linear problems by introducing kernel functions $k(\mathbf{x}_i, \mathbf{x}_j)$ into the optimization problem

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\Psi}(\boldsymbol{\alpha}) = \min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j k(\mathbf{x}_i \mathbf{x}_j) \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j - \sum_{i=1}^{N} \boldsymbol{\alpha}_i, \tag{12}$$

The kernel function transform our datapoints into a space where the datapoints are linearly separable.

Until here we considered decision boundaries that perfectly separate the classes. In the case of outliers this can however lead to over-fitting of the hyperplane to the training data. We therefore introduce the possibility to misclassify datapoints for the sake of generality introducing a penalty for misclassified datapoints that increases with distance to the decision hyperplane. Therefore slack variables ξ_i are introduced with $\xi_i = 0$ for correctly classified datapoints and $\xi_i = |u_i - y(\mathbf{x}_i)|$ otherwise.

Furthermore, to have control over the effect of slack variables we can further introduce a parameter C that trades off between the effect of slack variables and margin. We can show that the dual optimization problem is still represented by equation 12 with the additional constraint on $\boldsymbol{\alpha}_i$, i.e.,

$$0 \leq \boldsymbol{\alpha}_i \leq C. \tag{13}$$

The parameter C is typically found using a cross-validation procedure.

4.1 Assignments

For this assignment, we aim to explore the capabilities of SVMs on our classification data. We therefore explore the effects of different kernels, cross validation and posterior probabilities. Luckily, you will not have to implement the SVM mechanics yourself as the `scikit` package supplies implementations of the algorithms. In order to create an SVM classifier and train it on some input data, a simple program looks as follows

```
import numpy as np
from sklearn import svm

#features and labels are your n x d and n x 1 dimensional inputs.
classifier = svm.SVC(kernel='linear')
classifier.fit(features, labels)
```

In addition to the kernel type you can specify several more arguments, such as the penalty `C`, degree of polynomial for polynomial kernels, and several others. You can find a complete list and many examples in the documentation at `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. Note that we also have the option to enable probability estimates for the classification. You may wonder how this is possible, considering that we are dealing with binary classification tasks. While it is outside of the scope of this exercise to explain the details of this extension, you can find the derivations in [3] and refer to the lecture notes.

Tasks: Follow the steps of the repetition to complete the sections indicated with `#TODO` in `svm_main.py`. **Note: For this exercise you should load the provided feature and label data and not re-use the data of the previous exercises! Furthermore, do not train on your evaluation data!**

1. Normalize your feature data.
2. Implement an SVM with linear kernel that perfectly fits your input data. Apply the classifier to the evaluation data and store the result to `results_svm_Y_linear.txt`. For visualization purposes, you can retrain the classifier on the first 2 feature dimensions. However, make sure to save your results on all 5 feature dimensions.

3. Implement an SVM with Gaussian kernel that perfectly fits your input data. Apply the classifier to the evaluation data and store the result to `results_svm_Y_g_perfect.txt`. For visualization purposes, you can retrain the classifier on the first 2 feature dimensions. However, make sure to save your results on all 5 feature dimensions.

4. Implement an SVM with Gaussian Kernel that allows misclassifications while softly penalizing them with `C=1` and train it on your input data. Apply the classifier to the evaluation data and store the result to `results_svm_Y_g_slack.txt`. For visualization purposes, you can retrain the classifier on the first 2 feature dimensions. However, make sure to save your results on all 5 feature dimensions.

5. Retrain all three SVMs to also return probability estimates. Store the classification probabilities on the evaluation data to file, i.e.,

   (a) `results_svm_P_linear.txt`

   (b) `results_svm_P_g_perfect.txt`

   (c) `results_svm_P_g_slack.txt`.

   For visualization purposes, you can retrain the classifiers on the first 2 feature dimensions. However, make sure to save your results on all 5 feature dimensions.

The program will plot the classifications for the 2-dimensional case for you. You should expect a result similar to Fig. 4. Here, the plots are made with probability estimates. Expect sharp decision boundaries in the plots for the non probabilistic case.

5 Submission

To hand in the exercise you have to zip the whole folder `(4_0_pca_kmeans_svm)` with your code and result files and upload it on moodle. The zip file you upload should have the name *aifr_lastname_ex4* (replace *lastname* with your last name). The python scripts are written in a way, such that it always creates a `results*.txt` or `results*.pkl` file with the data that is used

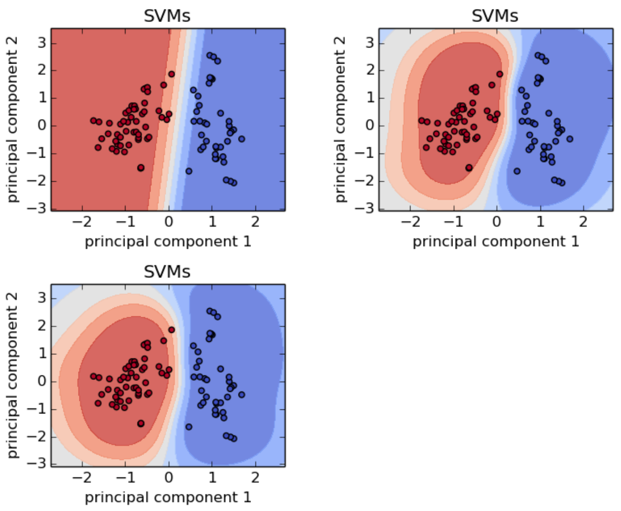

Figure 4: Expected result of SVMs with probability estimates.

for evaluation. **Please make sure that your python scripts are executable**.

References

[1] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, pp. 1–58, 2006.

[2] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[3] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.