

1 Data Preparation

In this first step, I was asked to prepare the images for the elaboration. The dataset was made by 1087 samples of 3x227x227 size which belong to 4 visual object categories. I extracted all the files in PACS_homework folder in the root of my workspace, then I used 4 threads (one for every category) each one running algorithm 1. I decided to save the images and the labels in two file (data.npy and label.npy) in order to make next loadings faster.

Algorithm 1: Projection applying PCA

```
62 def add_data_to_matrix(list_to_fill, pacs_dir, folder_name, color):
63     """
64     Thread function which fills a list with images as flattened arrays
65     :param list_to_fill: list to fill
66     :param pacs_dir: directory of PACS_homework
67     :param folder_name: folder/category name of images
68     :param color: label number
69     :return: None
70     """
71     print('Loading ' + folder_name + '...')
72     path_dir = pacs_dir + folder_name
73     directory = os.fsencode(path_dir)
74     lst = os.listdir(directory)
75     lst.sort()
76     for file in lst:
77         filename = os.fsdecode(file)
78         if filename.endswith(".jpg"):
79             img_data = np.asarray(Image.open(os.path.join(path_dir, filename)))
80             list_to_fill[0] = np.vstack((list_to_fill[0], img_data.ravel()))
81             list_to_fill[1] = np.vstack((list_to_fill[1], color))
82             # print(filename)
83             continue
84         else:
85             continue
86     print(folder_name + ' loaded correctly!')
```

2 Principal Component Visualization

I standardized x using `StandardScaler`, making each feature zero-mean and unit-variance. Then I applied PCA on the normalized x obtaining the projection using all 1087 PCs, as shown in algorithm 2 on the next page. I decided to implement a custom function called `reconstruction()` in order to re-project x_t using first 60, 6, 2 and last 6 principal components (PC) and visualize the reconstructed images with `show_reconstruction()` function, as shown in algorithm 3 on the following page.

Algorithm 2: Applying PCA of sklearn

```
249 # 1.2 PRINCIPAL COMPONENT VISUALIZATION
250 # Standardize the features
251 st = StandardScaler()
252 st.fit(x)
253 x_n = st.transform(x)
254 pca = PCA()
255 pca.fit(x_n)
256 x_t = pca.transform(x_n)
```

Algorithm 3: Re-projection on a specific range of components

```
142 def reconstruction(x_t, st, n_comp, comps):
143     """
144     Re-projection with n_comp
145     :param x_t: Projection normalized
146     :param st: Standard Scaler
147     :param n_comp: number of components (negative for last n_comp components)
148     :param comps: components
149     :return: re-projected image
150     """
151     if n_comp > 0:
152         x_b = np.dot(x_t[:, :n_comp], comps[:n_comp, :])
153     else:
154         # We want the last non-trivial components
155         x_b = np.dot(x_t[:, n_comp:], comps[n_comp:, :])
156
157     orig = st.inverse_transform(x_b) / 255
158     return orig.astype(np.float64)
159
160
161 def show_reconstruction(ax, x_t, st, pca: PCA, pca_n, image_id):
162     """
163     Plot image reconstruction specifying the number of PC
164     :param ax: axes to plot
165     :param x_t: Projection
166     :param st: Standar Scaler
167     :param pca: pca
168     :param pca_n: number of components (negative for last n_comp components)
169     :param image_id: id of the image to re-project
170     :return: None
171     """
172     if pca_n > 0:
173         var = float(np.sum(pca.explained_variance_ratio_[:pca_n]) * 100)
174         ax.set_title("%dPC var: %2.2f%%" % (pca_n, var))
175     else:
176         var = float(np.sum(pca.explained_variance_ratio_[pca_n:]) * 100)
177         ax.set_title("Last %dPC var: %2.2f%%" % (pca_n * -1, var))
178     x_pc = reconstruction(x_t, st, pca_n, pca.components_)
179     ax.axis('off')
180     ax.imshow(x_pc[image_id].reshape(227, 227, 3), vmin=0, vmax=1)
```

2.1 Comments about re-projections

I tried to re-project a series of images from different categories and I obtained the results in figs. 1 to 4 on pages 3–5.

Some coarse details of the original images can be detected in the re-projections with 60PC, because these have a cumulative variance of approximately 77%, but 6PC and 2PC plots has lower variances because of the reduced number of PC used to re-project.

The "Last 6PC" one has very low variance (approximately 0%), instead. Clearly, this result is due to PCA transformation: PCs are ranked according to variance (descending order), so last PCs could not be able to describe correctly the whole dataset.

It is easy to notice that all "Last 6PC" plots describe the shape of a person: this is probably because there is a different amount of images for each category and people pictures are the majority. Indeed, the dataset is composed by 189 dogs ($\approx 17\%$), 186 guitars ($\approx 17\%$), 280 houses ($\approx 26\%$) and 432 people ($\approx 40\%$).

Furthermore, the great part of people images are from the same point of view (frontal) in contrast to the other ones where there are different perspective, numbers and rotation: maybe this could be another underlying cause.

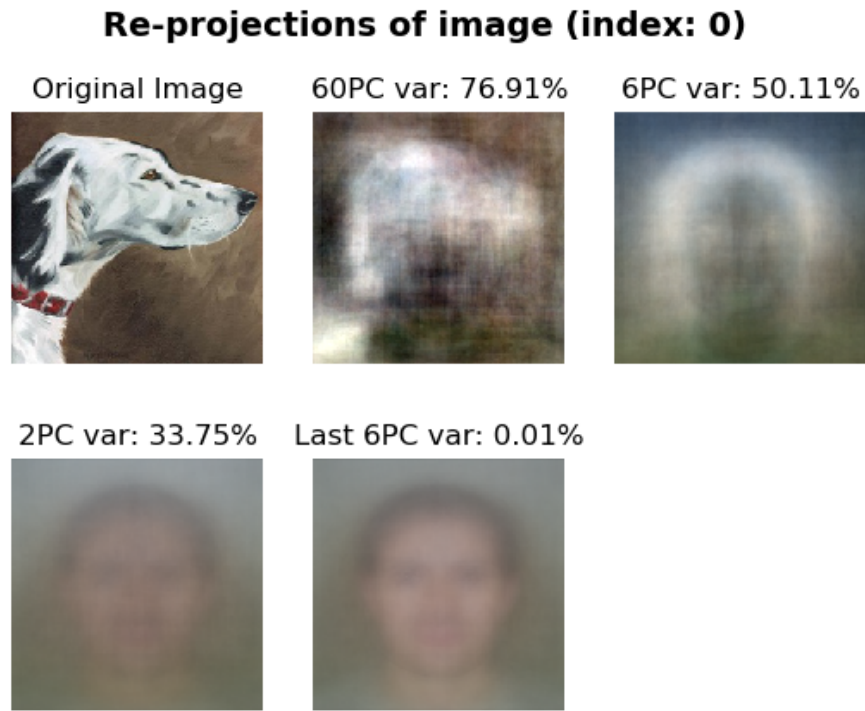


Figure 1: Re-projections of a *dog* image

Re-projections of image (index: 189)

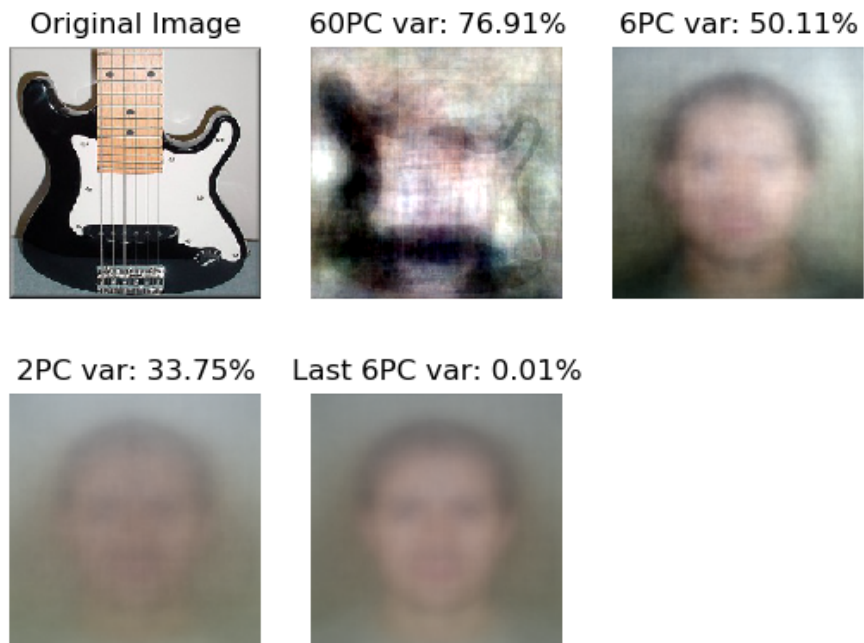


Figure 2: Re-projections of a *guitar* image

Re-projections of image (index: 375)

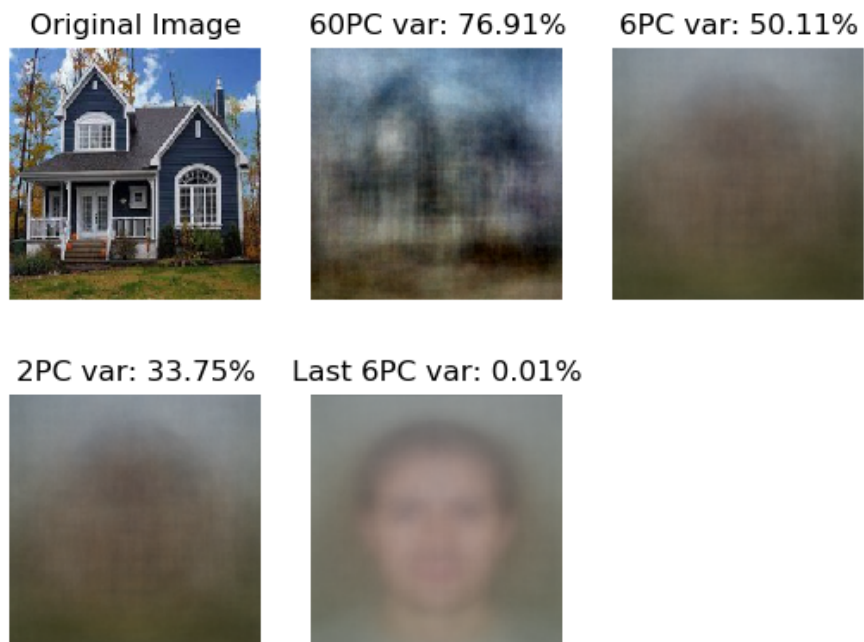


Figure 3: Re-projections of a *house* image

Re-projections of image (index: 677)

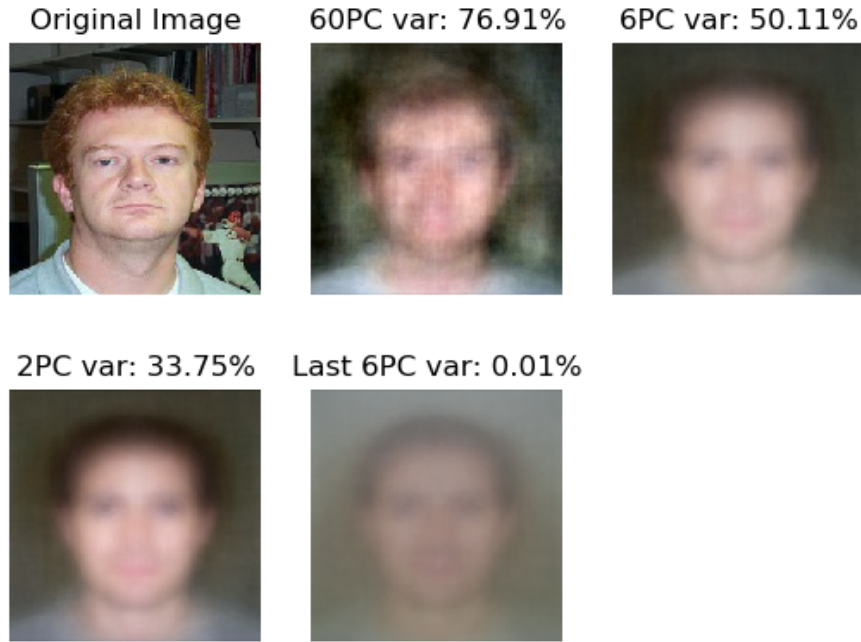


Figure 4: Re-projections of a *person* image

2.2 Comments about scatter and variance plots

I produced scatter plot using different combination of PCs (see figs. 5 to 8 on pages 6–7).

In fig. 5 there is a projection on 1st and 2nd PC: it is easy to find an area for *person* and *house* classes, while *guitar* and *dog* ones have a wider distribution on the plot.

In figs. 6 and 7 there are a 3rd and 4th PC projection and a 10th and 11th PC one: in this case it seems more difficult to find crowded and almost isolated areas of a single category.

It is noticeable that the range of coordinates in the graphs was about inversely proportional to the position of PC used, or, better, that the points in the scatter plot were getting closer and closer to the mean value (which is 0 because of normalization), causing a **decrease in variance**. This can be explained from theoretical perspective because, in the definition of PCA transformation, the first principal component has the **largest variance**, and each succeeding component in turn has the highest variance possible under the constraint that it is **orthogonal** to the preceding components.

Plotting the cumulative sum of variance ratio (provided in the PCA of `sklearn`) may be useful in order to decide how many components are necessary to preserve data without so many distortions. In general, there is not a right and unique way to select the *correct* number of principal components, but in this context we can analyze the graph in fig. 9 on page 8 and select a number of PC characterized by a cumulative sum of variance ratio higher than a specific threshold (e.g., 95%).

1° and 2° PC Scatter Plot of the Dataset

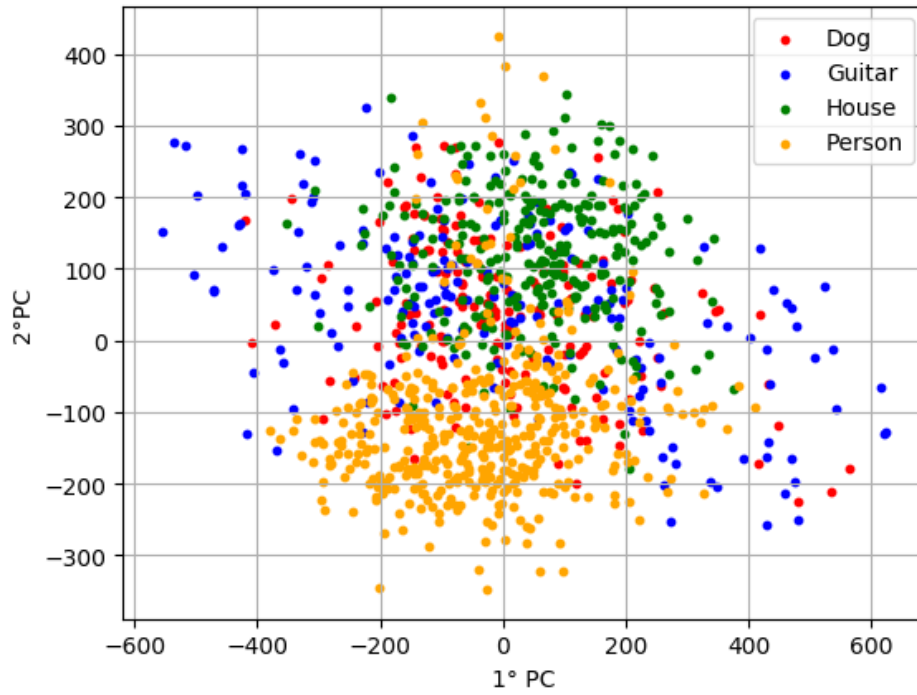


Figure 5: Projection on 1st and 2nd PC

3° and 4° PC Scatter Plot of the Dataset

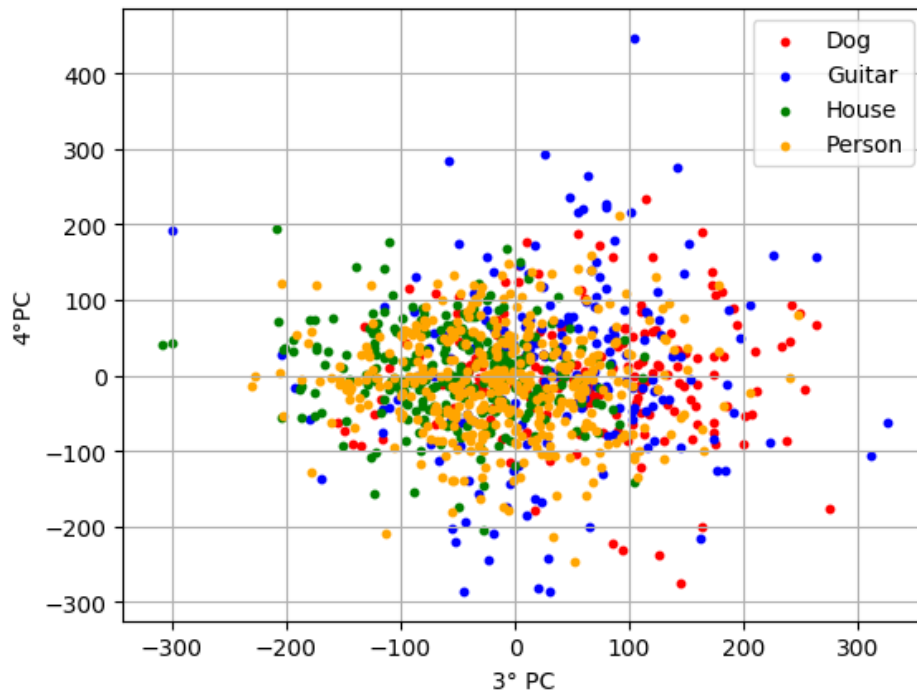


Figure 6: Projection on 3rd and 4th PC

10° and 11° PC Scatter Plot of the Dataset

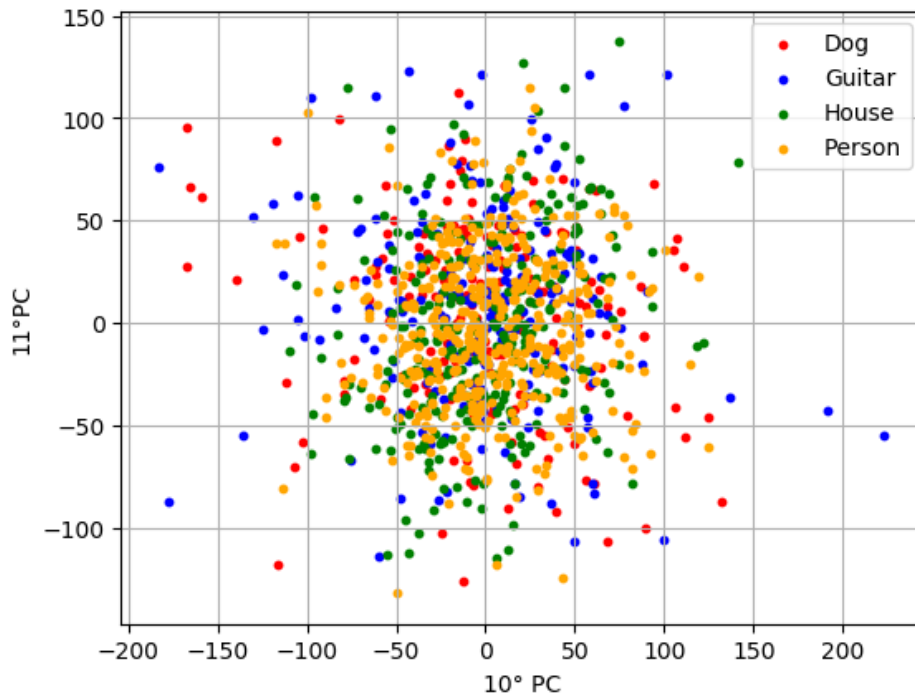


Figure 7: Projection on 10th and 11th PC

1°, 2° and 3° PC Scatter Plot of the Dataset

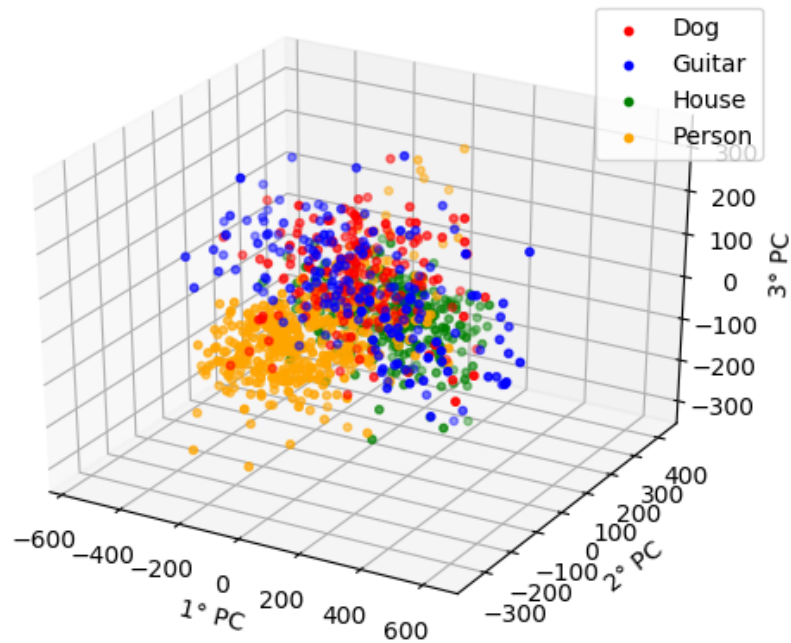


Figure 8: Projection on 1st, 2nd and 3rd PC

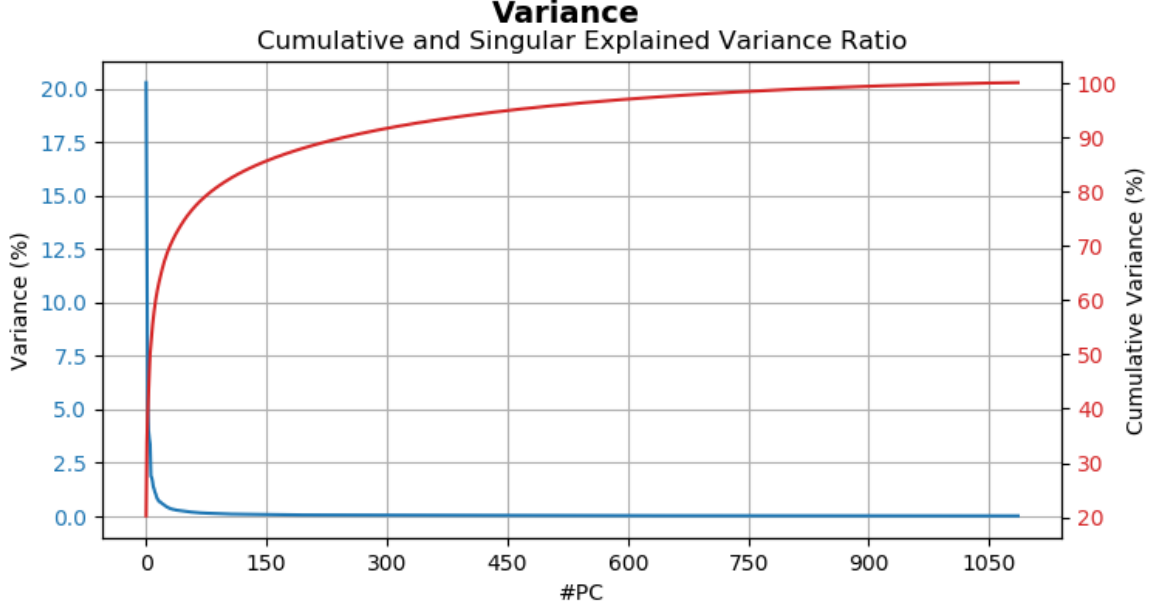


Figure 9: Variance and Cumulative Variance

3 Classification

The formulation of Naïve Bayes is described in eq. (1).

$$\hat{y} = \underset{i \in \{1, \dots, k\}}{\operatorname{argmax}} p(y_i | x_1, \dots, x_d) = \underset{i \in \{1, \dots, k\}}{\operatorname{argmax}} \underbrace{p(y_i)}^{\text{Prior}} \prod_{j=1}^d \underbrace{p(x_j | y_i)}^{\text{Likelihood}} \quad (1)$$

where

- \hat{y}_i = predicted label
- y_i = i -th label with $i \in \{1, \dots, k\}$
- x_j = j -th example with $j \in \{1, \dots, d\}$
- $p(x | y)$ = Gaussian

Firstly, I splitted randomly **x_n** (**x** standardized) and the labels in **training** (75%) and **test set** (25%), then I applied **GaussianNB** of **sklearn** obtaining an accuracy of **75%**.

After that, I splitted in the same way **x_t** (the projection of **x_n** obtained from PCA transformation) and I used these sets to train and test the classifier, first using the data projected on 1st and 2nd PC and finally the one projected on 3rd and 4th PC. In the first case the accuracy was **59.19%**, while in the second one it was **48.53%**.

I plotted **decision boundaries** of the classifier and the projection of the whole dataset for both cases, as we can see in figs. 10 and 11 on page 10.

Comparing accuracy results, I is clear that classifications done after the application of PCA are worse than the one applied on the original normalized dataset. Moreover, the accuracy of classifier on 1st and 2nd PC is higher than the one of classifier on 3rd and 4th PC, probably because of the different variance that a specific component can describe. This may be due to the fact that reducing dimensionality may discard important information useful to discriminate a class from the others in a classifier.

Algorithm 4: Classification and Plot of Decision boundaries

```

183 def plot_decision_boundaries_gnb(x_t, y, r_s, r_e, x_train, y_train, x_test, y_test,
184     title, legend_data, legend_color):
185     """
186     Useful to plot decision boundaries of Gaussian Naive Bayes Classifier
187     :param x_t: projection
188     :param y: labels
189     :param r_s: start of range of components
190     :param r_e: end of range of components
191     :param x_train: training data
192     :param y_train: training label
193     :param x_test: test data
194     :param y_test: test label
195     :param title: title to plot
196     :param legend_data: legend data
197     :param legend_color: legend color
198     :return: None
199     """
200     clf = GaussianNB()
201     clf.fit(x_train[:, r_s:r_e], y_train)
202     x_min, x_max = x_t[:, r_s].min() - 1, x_t[:, r_s].max() + 1
203     y_min, y_max = x_t[:, (r_e - 1)].min() - 1, x_t[:, (r_e - 1)].max() + 1
204     xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.5),
205                          np.arange(y_min, y_max, 0.5))
206
207     fig, axarr = plt.subplots(1, 1)
208     z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
209     z = z.reshape(xx.shape)
210
211     norm = matplotlib.colors.Normalize(vmin=0, vmax=3)
212     cmap_plot = matplotlib.colors.ListedColormap(legend_color)
213
214     axarr.contourf(xx, yy, z, alpha=0.5, norm=norm, cmap=cmap_plot)
215     for i in range(0, 4):
216         axarr.scatter(x_t[y == i, r_s], x_t[y == i, (r_e - 1)], c=legend_color[i],
217                      label=legend_data[i], s=20)
218     fig.suptitle(title, fontsize=14, fontweight='bold')
219     axarr.set_title(
220         "Accuracy over (%d points): %2.2f%%" % (x_test.shape[0], clf.score(x_test[:,
221         r_s:r_e], y_test) * 100))
222     axarr.set_xlabel("%do PC" % (r_s + 1))
223     axarr.set_ylabel("%do PC" % (r_e - 1))
224     axarr.legend()
225     fig.show()
226     print(title)
227     print("- Accuracy over (%d points): %2.2f%%" % (x_test.shape[0],
228         clf.score(x_test[:, r_s:r_e], y_test) * 100))

```

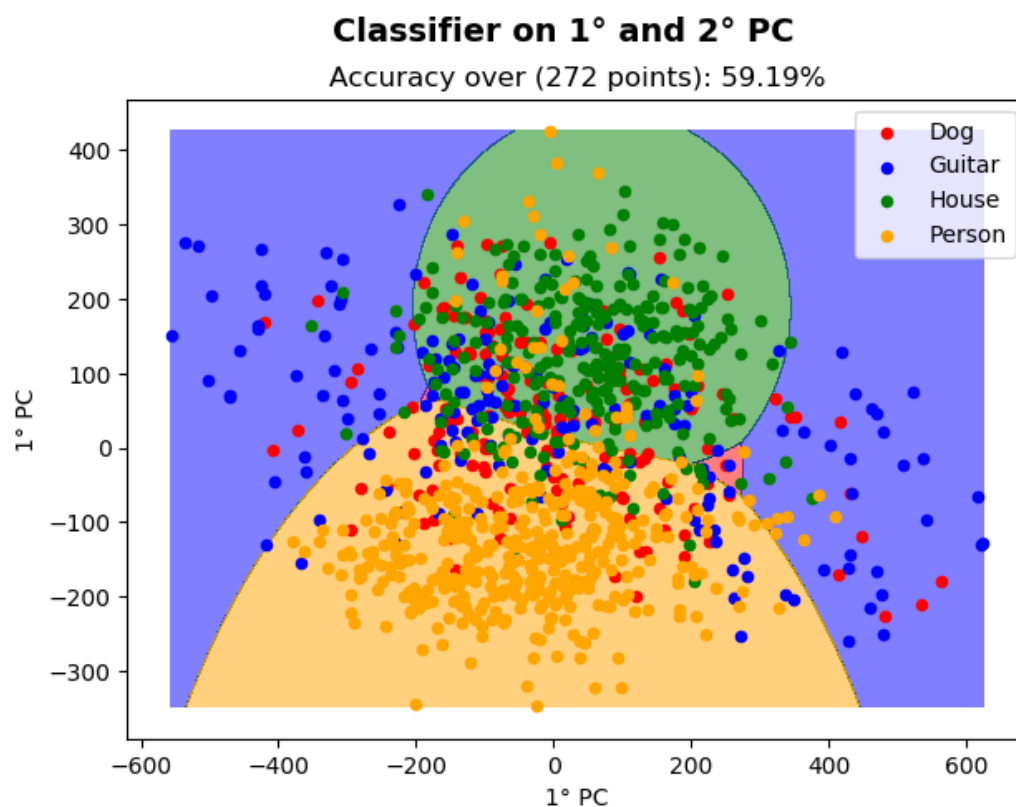


Figure 10: Decision boundaries of first classifier on 1st and 2nd PC

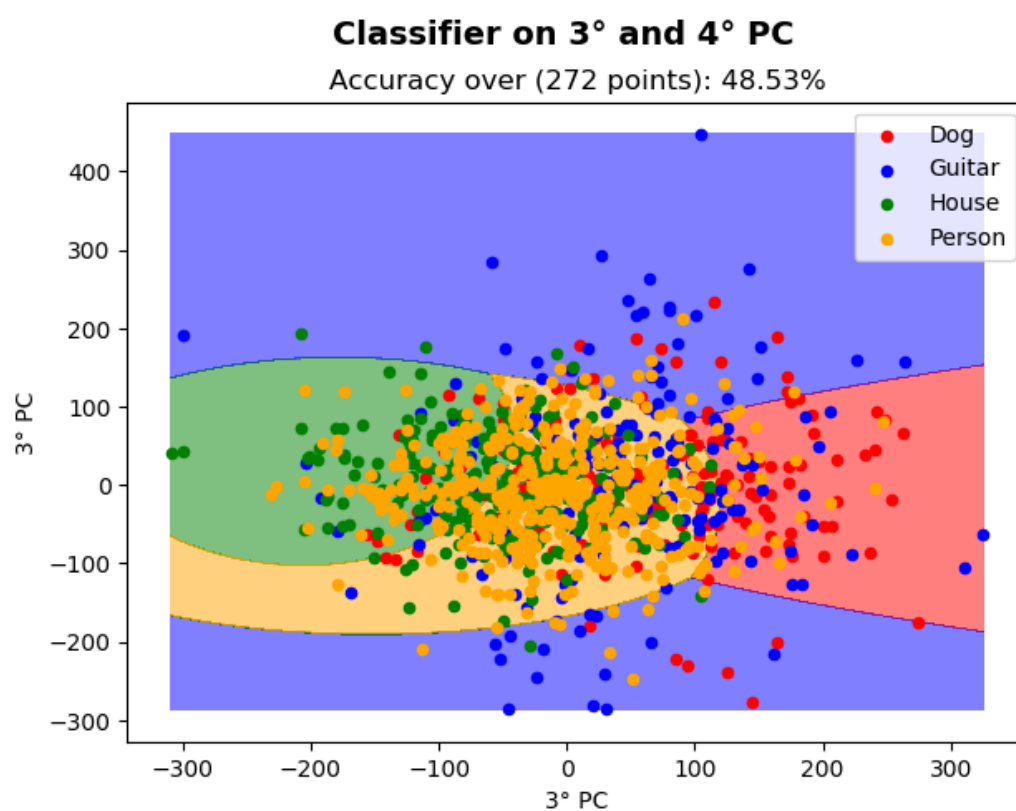


Figure 11: Decision boundaries of classifier on 3rd and 4th PC

4 Code Execution

4.1 Requirements

- Python 3
- All dependencies in `requirements.txt`.
\$ `pip install -r requirements.txt` to install them

4.2 Usage

- \$ `python main.py -n <PACS_homework folder>`
Loads Image from the specified `<PACS_homework folder>` and execute the code
- \$ `python main.py -s <PACS_homework folder>`
Loads Image from the specified `<PACS_homework folder>` and save files in `data.npy` and `label.npy` for faster next execution before executing the code
- \$ `python main.py -l <data.npy file> <label.npy file>`
Loads Image from `data.npy` and `label.npy` files and execute the code

4.3 Reproducibility

In order to reproduce the same data for this experiment you have to change the global variable `r_state` (line 25) from `None` to 252894 which is my badge number.

Attachments

- `source_code` folder:
 - `main.py`
 - `requirements.txt`