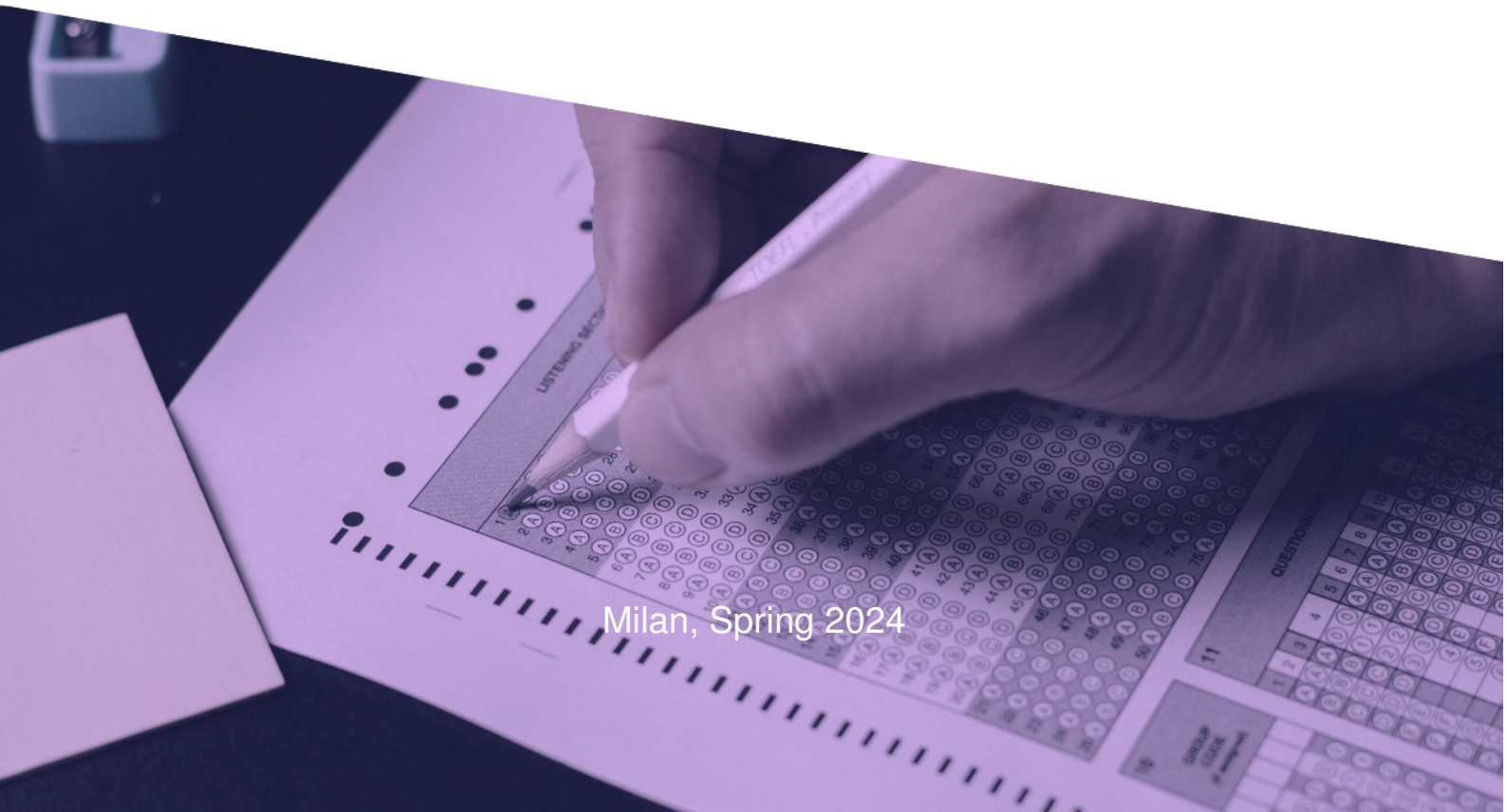
Automatic Quiz Scoring

Modernizing test feedback using Machine Learning

Hephaestus Applied Artificial Intelligence Association

Authors:

Member	Role
Samuele Nicolò Straccialini	Head
Sofia Guidotti	Member
Francesca Casillo	Member
Velina Todorova	Member



Milan, Spring 2024

Contents

1 Introduction **1**
- 1.1 Description of the project . . . 1
- 1.2 Roadmap . . . 1
- 1.3 Code . . . 1

2 Data **2**
- 2.1 Datasets . . . 2
- 2.2 Dataset cleaning . . . 2
- 2.3 Obtaining relevant content . . . 3
  - 2.3.1 Cascade Classifier . . . 3
- 2.4 Letters Extraction and Labelling . . . 3
- 2.5 Data Analysis . . . 4

3 Dimensionality Reduction **5**
- 3.1 Principal Component Analysis (PCA) . . . 5
  - 3.1.1 Maximum Variance . . . 5
  - 3.1.2 PCA steps . . . 6
- 3.2 PCA applied . . . 7

4 Supervised Learning **8**
- 4.1 Support Vector Machine . . . 8
- 4.2 Stochastic Gradient Descent Classifiers . . . 8
- 4.3 Note on GridSearchCV . . . 9
  - 4.3.1 SVM Classifier . . . 9
  - 4.3.2 SGD Classifier . . . 9

5 Conclusions and Further Research **10**

6 References **11**

1 | Introduction

1.1 | Description of the project

The objective of this project is to provide a comprehensive statistical analysis of responses collecteds from the "Campionati della Cultura e del Talento", an annual competition designed for Italian high school students. This contest involves a series of 50 multiple-choice questions, covering a range of topics that vary each year. Following the administration of the quiz, teachers grade the paper answer sheets and forward the scores along with scanned copies of the completed quizzes to the hosting association. Traditionally, the association selects a random subset of these quizzes for detailed analysis to extract data and insights.

Conversely, this project aims to enhance the scope of analysis by including all submitted scores. To achieve this, a program equipped with advanced letter recognition capabilities was developed, facilitating the precise classification and organisation of data into a well-structured table. This method allows for a more thorough and accurate evaluation of the competition's outcomes.

1.2 | Roadmap

The roadmap of the project can be divided into the following key phases:

1. **Dataset Collection:** The raw quiz data consists of scanned answer sheets provided by schools. This data serves as the foundation for the project.
2. **Data Preprocessing:** In this phase, the system parses the scanned quizzes to identify the format and structure of the answer sheets. Tables need to be extracted and normalized into a consistent form for easier analysis.
3. **Model Training:** Initially, the *EMNIST* dataset, which contains images of handwritten letters and digits, was used to train a model for classifying quiz answers. However, the results were suboptimal because the nature of the handwritten answers in the quizzes (due to variations in size and shape) differed significantly from the *EMNIST* samples.
4. **Custom Dataset Preparation:** To improve performance, a new dataset, comprising actual quiz images, was prepared. A subset of images, referred to as the *restricted dataset*, was manually labeled to train a model more accurately tailored to the quiz structure.
5. **Model Deployment and Evaluation:** Once the model is trained, it will be deployed to process all the quizzes. The system's performance will be evaluated based on its ability to detect and classify the answers into the appropriate categories (A, B, C, D).

1.3 | Code

The entire codebase for our analysis, including prediction algorithms, exported model parameters, and all other project-related developments, is publicly accessible on our GitHub repository at the following url: https://github.com/Hephaestus-AI-Association/Automatic-Quiz-Scoring. Please refer to the README files in the repository for instructions on how to use the code.

2 | Data

2.1 | Datasets

The project utilizes an image dataset that was adapted based on needs:

- **Raw Dataset:** This dataset consists of unprocessed scanned answer sheets submitted by schools. It contains tables where students have marked their answers. Variations in handwriting and potential noise from the scanning process present challenges for analysis.
- **Preprocessed Dataset:** To streamline the transition from the raw dataset to a cleaner dataset, a conversion process was implemented. A script was used to standardize images by converting various formats (PNG, JPG, PDF) into JPEG, ensuring compatibility for further processing. For PDF files, each page was extracted as a separate image. A PDF-to-image conversion library was utilized to accurately extract images from multi-page PDFs. By converting all files into a uniform format, the data became easier to process, enabling the creation of a clean, reduced dataset for model training.

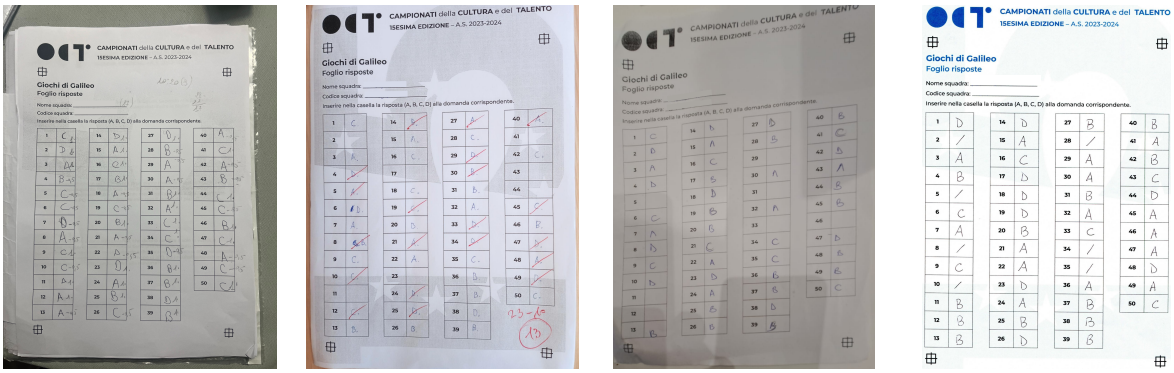


Figure 2.1: Some examples of pictures contained in the original raw dataset

2.2 | Dataset cleaning

Given the variations and noise present in the scanned documents, the dataset cleaning process includes several key steps to prepare the images for automatic quiz scoring such as:

- **Noise Reduction:** Filtering techniques are applied to raw images to enhance clarity. A bilateral filter is applied to reduce noise while preserving the edges of the text and table structures. This step is critical for accurate contour detection, ensuring that the essential features of the quiz, such as table cells, remain sharp.
- **Grayscale Conversion and Thresholding:** The images are then converted to grayscale to simplify further processing. Adaptive thresholding is applied to convert the grayscale images into binary format, creating a high contrast between the background and the table lines or marks. This enhances the accuracy of structure detection.

The last step was extracting the letter corresponding to each answer from each picture. Our first approach was directly applying a `opencv` built-in contour detection algorithm consisting in converting images to binary form and applying a *Canny edge detection* to them. However we encountered a major issue: not all detected contours represent table cells; by contrast, not all cells were correctly identified. We tried to filter out irrelevant contours based on their size and other relevant features. However, the results were cumbersome and not satisfactory enough. Consequently, we decided to adopt a different strategy which is described in the following sections.

2.3 | Obtaining relevant content

The first step to get our letters correctly cropped was working on a set of pictures with same size, orientation and characteristics. To have such a cleaned dataset we had to perform a further preprocessing action. Exploiting the presence of some **"target" symbols** (see Figure 2.2 for some examples) at the vertices of the answer table, we built a Cascade Classifier that was able to identify such symbols on each picture and crop it accordingly, as well as rotating and adjusting size and perspective if necessary. In the next section we describe briefly what is cascading and how such classifier works.

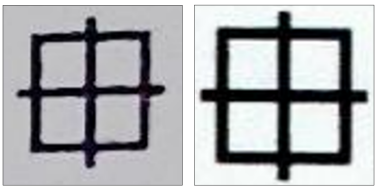


Figure 2.2: two examples of target symbols

2.3.1 | Cascade Classifier

Cascading is one of the most powerful ensemble learning algorithms.
It consists of multiple stages: the results collected as output of a given classifier are used as additional information for the next one. A Cascade Classifier is trained through several positive and negative examples of the target to learn the features that characterize (and don't characterize) the common object. In particular, we used the **Haar Cascade Classifier**, an object detection method proposed by Paul Viola and Michael Jones in 2001 [4]. Here's a simple explanation on how it works:

- Pictures are split into **integral images**, i.e. sub-rectangular regions. Haar features are then computed for each region. Since nearly all Haar features will be irrelevant when doing object detection (except the features of the object itself), this will speed up noticeably the calculation.
- **Haar features** are collected. Haar features are essentially calculations that are performed on adjacent rectangular regions in a detection window (integral image). The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. The resulting value indicate certain characteristics of the particular area, such as the existence (or absence) edges or changes in texture. For example, a 2-rectangle feature can indicate where the border lies between a dark region and a light region (look at Figure 2.3 for some examples).
- **Adaboost** (Adaptive Boosting) is applied to choose the the best features and to train the classifiers on them. It combines "weak classifiers" by sliding a window across the input image, calculating Haar features for each subsection, and comparing those values to a learned threshold that distinguishes objects from non-objects. These weak classifiers are then combined into a "strong classifier," which the algorithm uses for object detection.

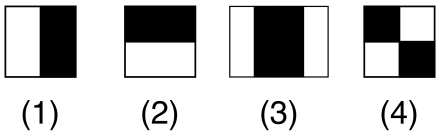


Figure 2.3: examples of 2-rectangle (1,2), 3-rectangle (3) and 4-rectangle (4) features.

The advantage of this technique compared to other Machine Learning algorithms is that, after training, the classifier can be applied to the entire image and detect the object in question in each region: in our case we could recognize multiple instances of the target symbols in each page. Also, notice that the Haar Classifier does not work on rotated figures, but the symmetric nature of the target symbol made us possible to use this algorithm anyway.

2.4 | Letters Extraction and Labelling

The next passage is letters extraction. Thanks to the process described in Section 2.3 we are able to crop the pictures and adjust them in order to have a standard output (i.e. answer tables of same size, shape and orientation). We can therefore extract all letters by simply **applying a mask**, which makes it possible to show only interesting areas. The Figure 2.4 below simply illustrates this simple process.
After extracting all the letters we need to label them in order to train our classifier on our handwritten dataset and test it afterwards. We did this process by hand.

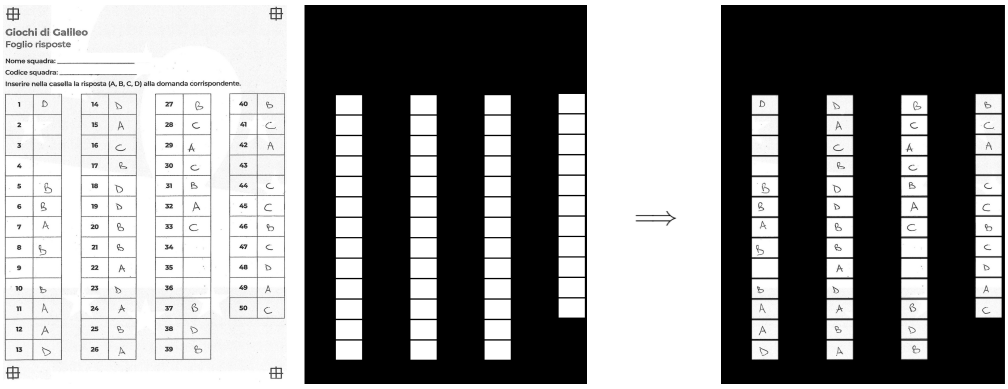


Figure 2.4: Example of answer table, mask and application of mask on table

2.5 | Data Analysis

We then performed some data analysis on our dataset. The table below shows the results of our labelling phase:

LABEL	COUNT
B	2529
A	2461
C	2430
D	1974
0	747
X	704
x	2
Z	1
E	1

The expected result was obtaining labels `A, B, C, D, 0` (empty cell) and `X` (unreadable cell); we therefore identified and corrected the labeling errors.

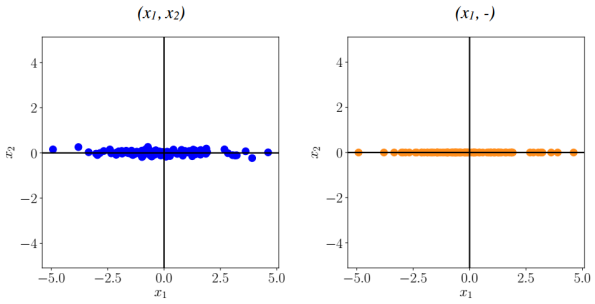
3 | Dimensionality Reduction

Analysing high-dimensional data comes with some difficulties: data is hard to analyze, interpretation is difficult, visualization is impossible and storage of data vectors is very expensive. However, we can exploit some properties to make the study more efficient:

- High-dimensional data is often over-complete, i.e., many dimensions are redundant and can be explained by a combination of other dimensions.
- Dimensions in high-dimensional data are often correlated (intrinsic lower-dimensional structure).

Dimensionality reduction exploits structure and correlation and allows us to work with a more compact representation of the data, ideally without losing information. We can think of dimensionality reduction as a compression technique, similar to jpeg or mp3, which are compression algorithms for images and music.

3.1 | Principal Component Analysis (PCA)

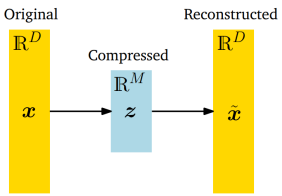


(a) Dataset with x_1 and x_2 coordinates.

(b) Compressed dataset where only the x_1 coordinate is relevant.

Figure 3.1: dimensionality reduction: the original dataset does not vary much along the x_2 direction, so we remove it obtaining nearly no loss.

One of the most widely used techniques for linear dimensionality reduction is Principal Component Analysis proposed by Pearson (1901) and Hotelling (1933).



Basically, we are interested in finding projections $\tilde{\boldsymbol{x}}_n$ of data points $\boldsymbol{x}_n$ that are as similar to the original data points as possible, but with a significantly lower intrinsic dimensionality. Consider the dataset $\mathcal{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ with $\boldsymbol{x}_n \in \mathbb{R}^D$ i.i.d. with mean 0 and variance-covariance matrix $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T$. We seek for a low dimensional representation of $\boldsymbol{x}_n$:

$$\boldsymbol{z}_n = \boldsymbol{B}^T \boldsymbol{x}_n \in \mathbb{R}^M \qquad (M < D)$$

$$\text{where:} \quad \boldsymbol{B} := [\boldsymbol{b}_1, ..., \boldsymbol{b}_M] \in \mathbb{R}^{D \times M}$$

The columns of $\boldsymbol{B}$ are orthonormal so that $\boldsymbol{b}_i^T \boldsymbol{b}_j = 0$ if and only if $i \neq j$ and $\boldsymbol{b}_i^T \boldsymbol{b}_i = 1$.

This means that first we seek an M-dimensional subspace $U \subseteq \mathbb{R}^D$ with $\dim(U) = M < D$ onto we project the data. We denote the projected data by $\tilde{\boldsymbol{x}}_n \in U$, and their coordinates (with basis vectors $\boldsymbol{b}_1, ..., \boldsymbol{b}_M$ of U) by $\boldsymbol{z}_n$. Our aim is to find $\boldsymbol{z}_n$ and the basis vectors $\boldsymbol{b}_1, ..., \boldsymbol{b}_M$ so that the projections $\tilde{\boldsymbol{x}}_n \in \mathbb{R}^D$ are similar to the original data $\boldsymbol{x}_n$: e.g. make $\|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2$ as small as possible.
The linear mapping represented by $\boldsymbol{B}$ can be thought of a **decoder**, which maps the low-dimensional code $\boldsymbol{z} \in \mathbb{R}_M$ back into the original data space $\mathbb{R}^D$. Similarly, $\boldsymbol{B}^T$ can be thought as an **encoder**, which encodes the original data $\boldsymbol{x}$ as a low-dimensional (compressed) code $\boldsymbol{z}$.

3.1.1 | Maximum Variance

Take a look again at Figure 3.1. We could have chosen to ignore the x_1-coordinate, but then the compressed data would have been quite uninformative with respect to the original data: in fact much information in the data would have been lost. The idea is to capture the information contained in the data by looking at the spread of it: directions of higher variance are more informative.

We start by seeking a single vector $\boldsymbol{b}_1 \in \mathbb{R}^D$ that maximizes the variance of the projected data, i.e. maximize the variance of the first coordinate $\boldsymbol{z}_1$ of $\boldsymbol{z} \in \mathbb{R}^M$:

$$z_{1n} = \boldsymbol{b}_1^T \boldsymbol{x}_n \qquad \text{projection of } \boldsymbol{x}_n \text{ in the direction } \boldsymbol{b}_1$$

$$V_1 := \mathbb{V}[z_1] = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{z}_{1n}^2 = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{b}_1^T \boldsymbol{x}_n)^2 = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{b}_1^T \boldsymbol{x}_n \boldsymbol{x}_n^T \boldsymbol{b}_1 = \boldsymbol{b}_1^T \left(\frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T \right) \boldsymbol{b}_1 =: \boldsymbol{b}_1^T \boldsymbol{S} \boldsymbol{b}_1$$

where $\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T$ is the data covariance matrix. Notice that V_1 increases with the magnitude of $\boldsymbol{b}_1$. However, we restrict all solutions to $\|\boldsymbol{b}_1\|^2 = 1$ and get the optimization problem:

$$\max_{\boldsymbol{b}_1} \boldsymbol{b}_1^T \boldsymbol{S} \boldsymbol{b}_1$$
$$\text{subject to } \|\boldsymbol{b}_1\|^2 = 1$$

The problem can be solved using the Lagrangian:

$$\mathcal{L}(\boldsymbol{b}_1, \lambda) = \boldsymbol{b}_1^T \boldsymbol{S} \boldsymbol{b}_1 + \lambda_1 (1 - \boldsymbol{b}_1^T \boldsymbol{b}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1} = 2\boldsymbol{b}_1^T \boldsymbol{S} - 2\lambda_1 \boldsymbol{b}_1^T = 0 \qquad \Longrightarrow \qquad \boldsymbol{S}\boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1$$
$$\frac{\partial \mathcal{L}}{\partial \lambda_1} = 1 - \boldsymbol{b}_1^T \boldsymbol{b}_1 = 0 \qquad \Longrightarrow \qquad \boldsymbol{b}_1^T \boldsymbol{b}_1 = 1$$

Then: $\boxed{V_1 = \boldsymbol{b}_1^T \boldsymbol{S} \boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1^T \boldsymbol{b}_1 = \lambda_1}$

We have seen how to find the first vector, which is the unit vector that maximizes the variance of the projected data points. However, in most applications, we want to find more than one direction: we look for the subsequent directions of high variance, that additionally must be orthogonal to the directions already considered in order to minimize redundancy in the information captured. Thus we define the k-th loading vector $\boldsymbol{b}_k$ as the solution to the constrained optimization problem:

$$\begin{aligned} &\max_{\boldsymbol{b}_k} \boldsymbol{b}_k^T \boldsymbol{S} \boldsymbol{b}_k \\ &\text{subject to} \quad \boldsymbol{b}_k^T \boldsymbol{b}_k = 1 \\ &\qquad\qquad\quad \boldsymbol{b}_k^T \boldsymbol{b}_i = 0 \text{ for } i = 1, ..., k-1 \end{aligned} \tag{3.1}$$

Similarly as before we get that $V_k = \lambda_k$ (where λ_k is the k-th larger eigenvalue) is the k-th maximal variance and the corresponding direction is given by the corresponding eigenvector (see [2] for more details).

3.1.2 | PCA steps

Having made such observations, the procedure goes as follows:

1. **Mean subtraction**: we center the data by computing the mean μ of the dataset and subtracting it from the data points. This passage is not strictly necessary but can be helpful.
2. **Standardization**: divide the data points by the standard deviation σ_d of the dataset for every dimension $d = 1, ..., D$.
3. **Eigendecomposition of the covariance matrix**: we compute the data covariance matrix, its eigenvalues and the corresponding eigenvectors. Since the covariance matrix is symmetric, the spectral theorem states that we can find an ONB of eigenvectors. The longer vector spans the principal subspace, which we denote by U.
4. **Projection**: we can project any data point $\boldsymbol{x}_* \in \mathbb{R}^D$ onto the principal subspace. We obtain the projection $\tilde{\boldsymbol{x}}_* = \boldsymbol{B}\boldsymbol{B}^T \boldsymbol{x}_*$ with coordinates $\boldsymbol{z}_* = \boldsymbol{B}^T \boldsymbol{x}_*$, where $\boldsymbol{B}$ is the matrix that contains the eigenvectors that are associated with the largest eigenvalues of the data covariance matrix as columns.

Going back to the original data space remember to undo the standardization by multiplying the standard deviation and adding the mean.

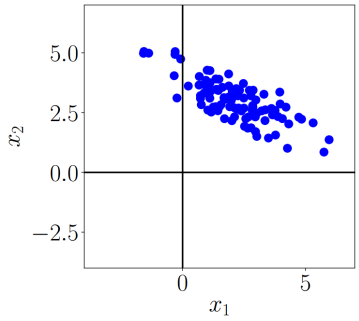


Figure 3.2: original dataset

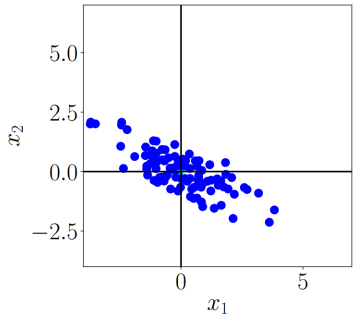


Figure 3.3: (step 1) mean subtraction

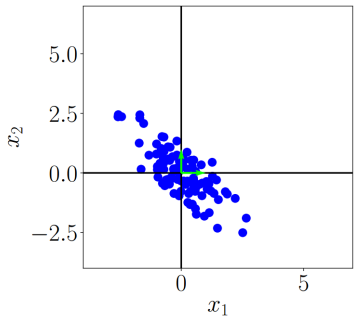


Figure 3.4: (step 2) standard deviation division

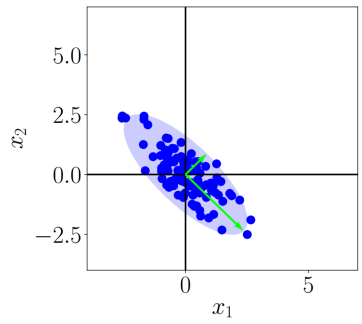


Figure 3.5: (step 3) compute eigenvalues and eigenvectors

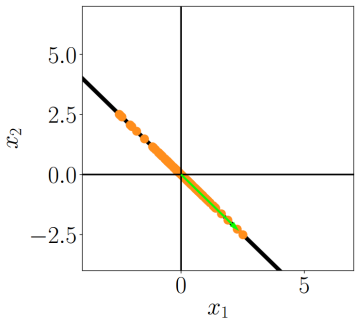


Figure 3.6: (step 4) project data onto the principal subspace

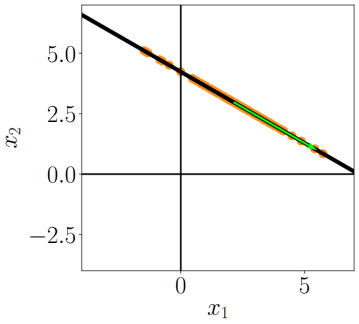


Figure 3.7: undo standardization and return back in the original data space

3.2 | PCA applied

PCA revealed to be an extremely useful tool for our scope. Working with a 11140×37440 dimensional array containing information on all single pixels for each image was indeed computationally consuming. Reducing our dataset to the first 400 principal components yielded the value of 95% explained variance (Figure 3.8). Below are also plotted datapoints when projected onto a 2 dimensional and 3 dimensional space. As expected from the low value of cumulative variance explained by the first two components (less than 50%) we cannot gain useful insights from these plots.

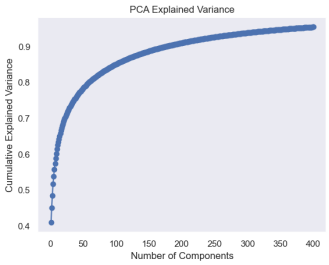


Figure 3.8: Cumulative Explained Variance per number of Principal Components

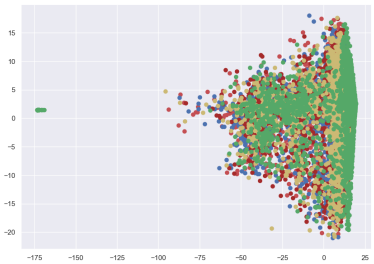


Figure 3.9: Datapoints plot after projecting in 2 dimensions

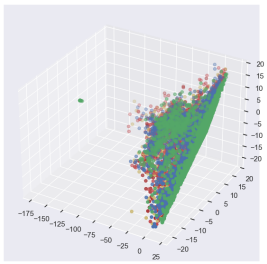


Figure 3.10: Datapoints plot after projecting in 3 dimensions

4 | Supervised Learning

4.1 | Support Vector Machine

The idea behind Support Vector Machine is to find an hyperplane that separates points of different classes by maximizing the distance between them [3].

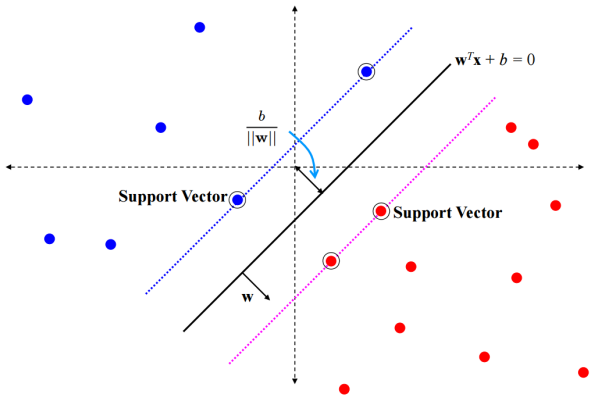


Figure 4.1: SVM in action in a 2D space

Recall an hyperplane in a D-dimensional $\mathbf{w}$ space has the form:

$$W \cdot x + b = 0 \quad \text{or} \quad W \cdot x = 0 \quad \text{where} \quad W_0 = b, x_0 = 1$$

and that the distance between a point $\boldsymbol{x}^\star$ and the hyperplane is:

$$d = \frac{|W \cdot \boldsymbol{x}^\star + b|}{\|W\|}$$

Since $\mathbf{w}^T\boldsymbol{x} + b = 0$ and $c(\mathbf{w}^T\boldsymbol{x} + b) = 0$ define the same plane, we have the freedom to normalize $\mathbf{w}$. We therefore proceed by normalizing the weight for the **positive and negative support vectors** respectively in the following way:

$$\mathbf{w}^T\boldsymbol{x}_+ + b = +1$$
$$\mathbf{w}^T\boldsymbol{x}_- + b = -1$$

The **margin**, i.e. the distance between the hyperplane and the observations closest to the hyperplane is given by:

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\boldsymbol{x}_+ - \boldsymbol{x}_-) = \frac{\mathbf{w}^T(\boldsymbol{x}_+ - \boldsymbol{x}_-)}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

Learning the SVM can be formulated as an optimization problem:

$$\max \frac{2}{\|\mathbf{w}\|} \quad \text{subject to} \quad \mathbf{w}^T\boldsymbol{x}_n + b \begin{cases} \geq +1 & \text{if } y_n = +1 \\ \leq -1 & \text{if } y_n = -1 \end{cases} \qquad \text{for } n = 1, ..., N$$

or equivalently:

$$\min \|\mathbf{w}\| \quad \text{subject to} \quad y_n(\mathbf{w}^T\boldsymbol{x}_n + b) \geq 1 \qquad \text{for } n = 1, ..., N$$

This is a quadratic optimization problem subject to linear constraints, and has a **unique minimum**. It can therefore be solved using Gradient Descent.

4.2 | Stochastic Gradient Descent Classifiers

Stochastic Gradient Descent is a widely used optimization algorithm, especially for large-scale learning tasks. It's an implementation of Gradient Descent which, instead of using the whole dataset, uses a randomly selected subset or single instance of data at each iteration. The idea behind the algorithm is to iteratively adjust the model parameters to minimize the error, given the objective function. The reason

why we choose to take a single instance of data rather then the whole dataset is because this reduces the computational work and therefore we get a faster convergence.

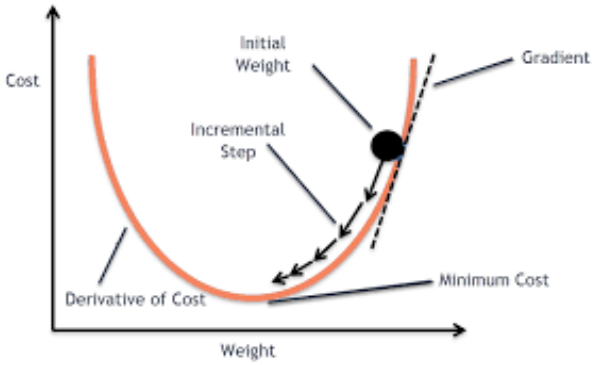


Figure 4.2: Example of SGD application

In SGD, we compute the gradient of the loss function with respect to the model parameters $\mathbf{w}$, so we update them in the opposite direction of the gradient to minimize the error:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla \mathcal{L}(\mathbf{w}^k)$$

In Python we use **SGDClassifier**, which is a linear classifier that optimizes the given empirical loss function using SGD.

4.3 | Note on GridSearchCV

GridSearchCV is a tool for hyperparameter tuning in machine learning models [1]. After specifying the hyperparameters, it performs an exhaustive search over a grid made of every possible combination of them. The evaluation is done using cross evaluation and the best combination is chosen based on the desired scoring metric. In the end, **GridSearchCV**outputs the best set of hyperparameters and the model trained with them.
N.B.: **GridSearchCV** spans over **all** the possible combinations of hyperparameters, which is good but can become computational expensive as the number of hyperparameters and their possible "values" increase.
We tuned different hyperparameters for both classifiers we tried.

4.3.1 | SVM Classifier

The hyperparameters tuned for **SVC**(SVM Classifier) are:

- **C**: controls the trade-off between low error on the training data and low model complexity. If the value of C is low(soft margin) the model generalizes better but allows more misclassifications. On the other hand, a high value for C(hard margin) has a lower tolerance for error but a higher risk of overfitting.
- **gamma**: quantifies the influence of single training examples. A low value leads to smooth decision boundaries, while a higher value can capture more details but, again, with higher risk of overfitting.
- **kernel**: defines the type of transformation applied to the data to find the optimal boundaries. It defines how the data is manipulated to find the hyperplanes that divide the classes.

4.3.2 | SGD Classifier

The hyperparameters tuned for **SGDClassifier** are:

- **loss**: specifies the loss function to be minimized in the training.
- **penalty**: specifies the regularization used to prevent overfitting.
- **alpha**: controls the weight of the penalty term in the loss function. A low value for alpha leads to weak regularization, with a model which is likely to overfit. A high value for alpha leads to a strong regularization, which forces a simpler model that might underfit.

5 | Conclusions and Further Research

The Support Vector Machine (SVM) classifier, after hyperparameter optimization, proved to be the best performing model, with an accuracy of approximately 80%. The precision and recall metrics demonstrate the classifier's effectiveness in identifying the correct answers across a majority of the test cases, particularly in identifying the 0 (empty cell) label, which had the highest accuracy. However, there is still room for improvement in the classification of letters such as D and B, which showed slightly lower accuracy. The table summarizes the various scores for each letter.:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	437
A	0.74	0.81	0.78	615
B	0.72	0.70	0.71	632
C	0.86	0.86	0.86	608
D	0.73	0.65	0.69	494
accuracy			0.79	2786
macro avg	0.80	0.80	0.80	2786
weighted avg	0.79	0.79	0.79	2786

The results are quite satisfactory: the ability to scale the model across the entire dataset offers significant benefits over the traditional method of random sampling. Indeed, we believe that possibility of handling the complete dataset (even with an accuracy of about 80%) would be more useful for statistical purposes than just sampling some tests.

Further researches might focus on several aspects. First, enhancing the dataset cleaning process could lead to improved classification performance. Experimenting with advanced techniques for noise reduction and letter extraction, such as using convolutional neural networks (CNNs) or other deep learning methods, might help mitigate the current limitations in accurately identifying less distinct characters. Finally, to enhance the versatility of our model, it could be possible to expand its capabilities to recognize a broader range of letters and symbols. In this way the algorithm would be capable to process tests that offer a wider array of possible responses.

6 | References

[1] Gridsearchcv. https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html.

[2] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for machine learning.* Cambridge University Press, 2020.

[3] A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 1998.

[4] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.