Eigenfaces for Recognition

Angelos Filos
Imperial College London
angelos.filos14@imperial.ac.uk

Youssef Rizk
Imperial College London
youssef.rizk14@imperial.ac.uk

Abstract

We present an approach to face recognition, using Principle Component Analysis (PCA) for dimensionality reduction and multi-class classifiers, such as Nearest Neighbours (NN) and Support Vector Machines (SVM). Extensive research is conducted to find the parameters that maximise recognition accuracy.[1]

1. Dataset & Definitions

A set of $N = 520$ face images, of size 46×56 is used, describing a set of $L = 52$ faces. It is normalised and centered, so no augmentations or affine transformations are used for data enrichment. Each image is flattened and represented as a column-vector $\mathbf{x}_i$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $D = 46 \times 56 = 2576$.

2. Eigenfaces

The number of available data $N = 520$ is significantly smaller than the number of features $D = 2576$ of each datapoint (image). Because of this sparsity we project the data to a smaller $\mathcal{M}-$dimensional subspace $\mathcal{F}$, where $\mathcal{M} < D$. With this transformation we aim to capture the essential information available in the provided data, represented in a compressed form, in order to improve computational complexity and remove redundancy.

2.1. Naive PCA ($\mathcal{S}_n = \frac{1}{N} AA^T$)

2.1.1 Data Preprocessing

The provided data are split into training and testing sets. A ratio of 4:1 training to testing data was selected after model evaluation and validation. We note specifically that we apply this division *class-wise* rather than to the whole dataset, to ensure equal representation of each label in both sets. Otherwise, classifiers are biased against the labels that are poorly represented in the training set. We denote with $N_{train} = 416$ and $N_{test} = 104$ the cardinalities of the two respective sets.

We normalise the training data before fitting any classifier by subtracting from every image the mean face $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$ (Figure 7), such that $\boldsymbol{\phi}_i = \mathbf{x}_i - \bar{\mathbf{x}}$ is the normalised i^{th} image from the training set. We subsequently define the matrix $A \in \mathbb{R}^{D \times N_{train}}$, whose columns are the normalized training images, i.e. $A = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, ..., \boldsymbol{\phi}_{N_{train}}]$.

2.1.2 Theoretical Results

The selection of the $\mathcal{F}$ subspace is done by picking the $\mathcal{M}$ eigenvectors of the covariance matrix $\mathcal{S}_n = \frac{1}{N} AA^T$, corresponding to the $\mathcal{M}$ greatest eigenvalues $\lambda_{1:\mathcal{M}}$ of $\mathcal{S}_n$. Since $\mathcal{S}_n$ is real and symmetric[2], its eigenvalues $\lambda(\mathcal{S}_n)$ are real [3] and thus they can be sorted. We can also argue for the number of non-zero eigenvalues of $\mathcal{S}_n$. Since $A \in \mathbb{R}^{D \times N_{train}}$:

$$rank_A \leq min(D, N_{train}) = N_{train}$$

According to identity [1] $rank_A = rank_{AA^T} = rank_{\mathcal{S}_n}$:

$$rank_{\mathcal{S}_n} \leq N_{train} = 416 \tag{1}$$

Therefore we expect $\mathcal{S}_n$ to have at most $D - N_{train}$ non-zero eigenvalues.[3]

2.1.3 Empirical Verification

We now proceed to empirically verify these results. When calculating the eigenvalues of $\mathcal{S}_n$, we actually obtain D non-zero eigenvalues, which seems to contradict the theory. However, $D - N_{train}$ of those eigenvalues are in the order of 10^{-11} and below, which is

[1] The code for all the experiments is available at https://github.com/filangel/Eigenfaces.

[2] $\mathcal{S}^T = (AA^T)^T = AA^T = \mathcal{S}_n$.

[3] We can further upper-bound $rank(A)$, since matrix A is the centered version of matrix X. According to [2]

$$rank(A) = rank(X) - 1$$

and then

$$rank(A) \leq N_{train} - 1 = 415$$

Accuracy (%)	Principle Components $\mathcal{M}$
80	26
90	65
95	121
99	265

Table 1. Reconstruction Accuracy - Number of Principle Components pairs.

essentially 0 with a precision error stemming from the numerical calculation of the eigenvalues. Thus, we do indeed obtain N_{train} non-zero eigenvalues. Moreover, we note that these eigenvalues are real, corroborating our theoretical hypotheses from subsection 2.1.2.

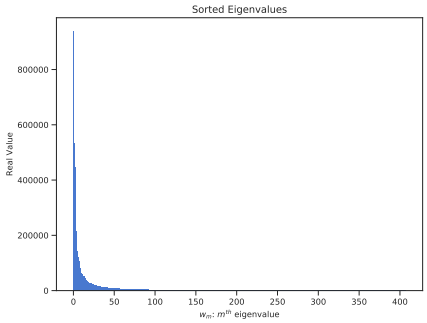

Figure 1. Eigenvalues sorted by magnitude

2.1.4 Thresholding

Using all N_{train} eigenvectors corresponding to the N_{train} non-zero eigenvalues would enable us to perfectly reconstruct the input images through a linear combination of these eigenvectors. However, as the eigenvalues in this context are representative of the relevance of the corresponding eigenvector to the reconstruction, we observe qualitatively from Figure 1, that only a fraction of the eigenvectors are playing a relevant part in the reconstruction. Thus, we could still obtain accurate reconstructions using a small subset of the eigenvectors of $\mathcal{S}_n$. Quantitatively, according to equation (7) we can control the reconstruction error by varying $\mathcal{M}$. Therefore we choose $\mathcal{M}$ such that we have 5% reconstruction error, by picking the $\mathcal{M}$ eigenvectors whose eigenvalues sum to 95% of the sum of all eigenvalues. In this case $\mathcal{M} = 121$. Table 2.1.4 further highlights the relationship between the reconstruction accuracy and the number of principle components pairs.

2.2. Efficient PCA ($\mathcal{S}_e = \frac{1}{N} A^T A$)

It is worth noting that $\mathcal{S}_n \in \mathbb{R}^{D \times D}$ is a large matrix and as a consequence it is computationally expensive to calculate its eigenvalue-eigenvector pairs. Exploiting the symmetry of $\mathcal{S}_n$ by using the result from (5), we infer the important (non-zero) eigenvalues and their corresponding eigenvectors of $\mathcal{S}_n$ from the eigenvalues and eigenvectors of a different matrix, $\mathcal{S}_e = \frac{1}{N} A^T A \in \mathbb{R}^{N_{train} \times N_{train}}$. This is a more practical approach since $N_{train} << D$ leading to identical results with a significant reduction in execution time, because of the $\mathcal{O}(n^3)$ time complexity of eigendecomposition of a matrix.

2.2.1 Theoretical Results

We now show that the eigenvalues and eigenvectors of $\mathcal{S}_n$ and $\mathcal{S}_e$ are related. Let λ and $\mathbf{v}$ be an eigenvalue-eigenvector pair of $\mathcal{S}_e$, then:

$$\mathcal{S}_e \mathbf{v} = \lambda \mathbf{v} \iff \frac{1}{N} A^T A \mathbf{v} = \lambda \mathbf{v} \tag{2}$$

Multiplying both sides with A:

$$\frac{1}{N} A A^T A \mathbf{v} = \lambda A \mathbf{v} \iff \mathcal{S}_n A \mathbf{v} = \lambda A \mathbf{v} \tag{3}$$

Let $\mathbf{u} = A\mathbf{v}$:

$$\mathcal{S}_n \mathbf{u} = \lambda \mathbf{u} \tag{4}$$

Therefore we proved that $\mathcal{S}_n$ and $\mathcal{S}_e$ have the same eigenvalues λ and their eigenvectors are related by:

$$\mathbf{u}_{\mathcal{S}_n} = A \mathbf{v}_{\mathcal{S}_e} \tag{5}$$

2.2.2 Empirical Verification

Building on the theoretical result of subsection 2.2.1, we now validate the theory with experiments. To prove the quantitative equivalence between the non-zero eigenvalues of the two matrices, we sum the absolute difference between their values for the largest N_{train} eigenvalues, and indeed, we note that this figure is 0. We proceed to verify that the eigenvectors (eigenfaces) are related by the aforementioned transformation (5). We confirm both numerical and qualitative equivalence, which is depicted in Figure 4.4.

2.2.3 Comparison

We choose to work with $\mathcal{S}_e$ through the rest of the report, because of its more practical computation and its both theoretical and empirical equivalence with $\mathcal{S}_n$. We note that the nature of the dataset is such that $N_{train} << D$ and due to the inequality (1) there can be

at most N_{train} linearly independent eigenfaces therefore the extra $D - N_{train}$ of $\mathcal{S}_n$ are redundant and there is no benefit calculating, storing and using them. Comparing the actual execution time of finding the eigenvalues and eigenvectors for the two cases, we obtained:

- Naive PCA $\mathcal{S}_n$: 25.14 seconds
- Efficient PCA $\mathcal{S}_e$: 0.89 seconds

3. Application of Eigenfaces

The following experiments are conducted using the **Efficient PCA** method described in section 2.2. The number of principle components $\mathcal{M}$ is treated as a hyperparameter, whose impact on the results is investigated in subsections 3.1.2 and 3.2.2.

3.1. Reconstruction

With PCA we have identified the $\mathcal{M}$ most significant features (**eigenfaces**) of the provided dataset. Additionally, we have projected the images from the input space $\mathcal{X}$ to the compressed feature space $\mathcal{F}$, spanned by the $\mathcal{M}$ mutually orthonormal eigenfaces. Now we investigate the inverse transformation: the reconstruction of an image from $\mathcal{F}$ to $\mathcal{X}$.

3.1.1 Algorithm

The reconstruction of the n^{th} compressed image in $\mathcal{F}$, represented by projection coefficients $w_n \in \mathbb{R}^{\mathcal{M}}$ is performed according to:

$$\tilde{x}_n = \bar{x} + \mathbf{U} w_n, \quad \tilde{x}_n \in \mathbb{R}^D \tag{6}$$

where:

- $\bar{x} \in \mathbb{R}^D$: the mean face of the training set
- $\mathbf{U} \in \mathbb{R}^{D \times \mathcal{M}}$: $\mathcal{M}$ (column-vector) eigenfaces

3.1.2 Results

For $\mathcal{M} < D$, PCA is a lossy and non-invertible operation (compression) that introduces reconstruction error $\mathcal{J}$ given by:

$$\mathcal{J} = \sum_{n=1}^{N} e_n = \sum_{n=1}^{N} \|x_n - \tilde{x}_n\| \propto \sum_{i=\mathcal{M}+1}^{D} \lambda_i \tag{7}$$

where λ_i is the i^{th} eigenvalue of the covariance matrix $\mathcal{S}$, sorted in a descending order.

Figure 2 illustrates the dependency of the reconstruction error $\mathcal{J}$ on number of principle components $\mathcal{M}$. Additionally, comparing Figures 2 and 1 we experimentally confirm the last step in equation (7) and justify our algorithm for selecting $\mathcal{M}$ in subsection 2.1.4.

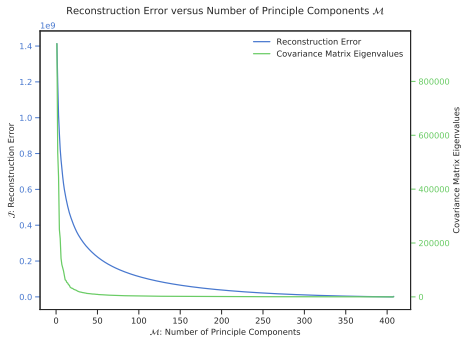

Figure 2. Reconstruction error $\mathcal{J}$ versus number of principle components $\mathcal{M}$.

Qualitative evaluation of the reconstruction error for both training and testing samples is depicted in Figure 4.4.

3.2. Nearest Neighbour (NN) Classification

The algorithm depends on two hyperparameters:

1. $\mathcal{M}$: the number of principle components for PCA
2. k: the number of nearest neighbours used for the classification

3.2.1 Algorithm

Assume a fitted PCA with:

- $\mathbf{U}$: the eigenfaces
- $\bar{x}$: the mean face of the training set
- $W \in \mathbb{R}^{\mathcal{M} \times N}$: compressed feature matrix of the training data, on the eigenface subspace $\mathcal{F}$
- $Y \in \mathbb{R}^N$: the labels of the training data

Algorithm 7 describes the **K Nearest Neighbours Classification** method. In step 2 we remove the mean face of the training set $\bar{x}$ from each test point x_i, under the assumption that all images are drawn from the same generative process with same statistics, therefore testing images have the same mean as the training set.

3.2.2 Results

In the case of $k = 1$ regardless of the value of $\mathcal{M}$, performance on the training set is always perfect, $accuracy = 100\%$, since the distance between a point and itself is always zero.

For a fixed value of $\mathcal{M}$, k is varied and the accuracy of the classifiers on the testing set is summarised in

Algorithm 1: k Nearest Neighbours Classification

foreach x_i in $\mathcal{D}_{test}$ **do**
  subtract the mean face $\bar{x}$ from x_i
  project normalised image to $\mathcal{F}$
  calculate Euclidean Distance in $\mathcal{F}$ from all training samples
  keep the labels of the k training samples with minimum distance
  use simple majority vote to determine the predicted label $y_{hat}(i)$
end

k	Recognition Accuracy (%)
1	60.71
3	42.86
5	42.86
7	37.51

Table 2. Recognition Accuracy against Number of Nearest Neighbors k.

Table 3.2.2. We note that the best model is the one with $k = 1$, because of the sparsity of the dataset.

Nonetheless, accuracy on the testing set is strongly dependent on $\mathcal{M}$, as expected from the error introduced during the PCA, covered in section 3.1. The accuracy is depicted as a function of $\mathcal{M}$ in Figure 3.

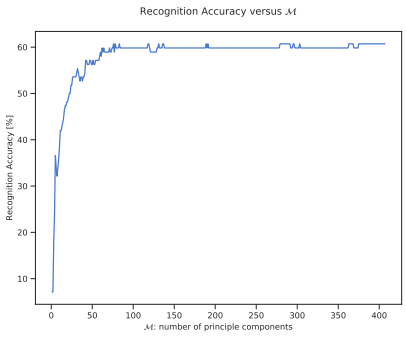

Figure 3. Nearest Neighbour Recognition Accuracy as a function of $\mathcal{M}$.

Higher values of $\mathcal{M}$ lead to improved accuracy, but this comes to a computational and storage trade-off, as depicted in Figure 4.4.[4] Examples of successful and failed classifications are also provided in the appendix, Figure 11.

Overall, recognition accuracy for $k = 1$ and $\mathcal{M} = 121$ was **60.7**% and the unnormalised confusion matrix is provided at Figure 4.

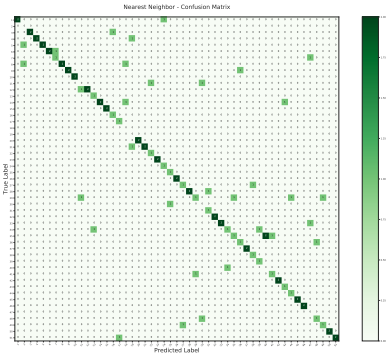

Figure 4. Nearest Neighbour Confusion Matrix.

4. Multi-class SVM for Face Recognition

In this section, we explore an alternative approach to classifying the given face images, **Support Vector Machines** (SVM). Vanilla SVMs are binary classifiers, therefore we use two modifications, namely *one-vs-all* (OVR) and *one-vs-one* (OVO), to extend them to multi-class classification. Both approaches are applied to the raw images, in space $\mathcal{X}$, as well as to compressed representations, in space $\mathcal{F}$ with varying number of principle components. Despite the small gain in accuracy by using features in $\mathcal{X}$, comparable performance was achieved in $\mathcal{F}$ with a significant computational benefit, see Table 4.4.

4.1. One-Versus-Rest

The `fit` (train) and `predict` (test) methods for OVR-SVM are provided in Listing 1. In training, L binary SVM classifiers are fitted, one for each label class. During testing, each image is fed to each of the L trained classifiers and the one that maximizes the probability score[5] is chosen.

The advantage of this method is its small number of classifiers, just L. On the other hand, these classifiers are trained on unbalanced data, such that only 8 samples are labelled using the positive class and the other 408 are labelled using the negative class.

4.2. One-Versus-One

The `fit` (train) and `predict` (test) methods for OVO-SVM are provided in Listing 2. In training, $L(L-1)/2$ binary SVM classifiers are fitted, one of each pair-combination of classes. During testing, each

[4]The experiments are run at `Google Cloud Platform` on a `Ubuntu 16.04.3 LTS` box, powered by `Intel(R) Xeon(R) CPU @ 2.30GHz` and `32GB DDRAM`.

[5]The probability score can be interpreted as the inverse of the distance of the point from the decision boundary.

image is fed to each of the $L(L-1)/2$ trained classifiers and the majority vote is taken for the true class.

The main advantage of this method is the simplicity of the binary classifiers, which are training only on a small number of points (16 in this example, 8 for each of the two classes), compared to the 416 points in case of OVR-SVM. However, the number of classifiers is growing quadratically with the number of class labels L.

4.3. Model Selection

Grid search 3-fold cross-validation is performed for each trained binary SVM, for the parameters:

- **C**: Penalty parameter of the error term
- **kernel type**: input transformation function
- γ: Kernel coefficient for `rbf` and `polynomial`
- **degree**: Polynomial degree for `polynomial`

As a general result, we found that **Linear SVM** outperformed all the other kernels, most likely because of the sparsity of the data and the resultant tendency for the Polynomial and especially RBF kernels to overfit the training set. Consequently, the parameters γ and **degree** do not affect the **Linear SVM** and thus we provide only the validation of **C** in Figure 4.3. We note in both cases, OVR and OVO, **C** is chosen to be large in order to regularize the classifiers. Especially for OVR we notice that the cross-validation recognition accuracy is maximized for very large values of **C**, order of 10^6. We also investigated the mean train and test time for the different kernel types and as expected **Linear SVM** was ten times faster in both train and test time compared to RBF kernel.[6]

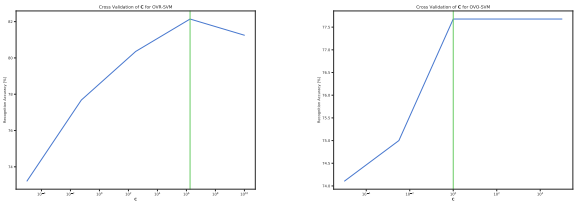

Figure 5. Recognition accuracy against Linear SVM penalty parameter **C**, for OVR (left) and OVO (right).

[6]OVR-SVM: linear mean fit time 0.008s vs. rbf mean fit time 0.074.

Method	Accuracy (%)
OVR-SVM	83.93
OVO-SVM	80.36
OVR-SVM & PCA ($\mathcal{M} = 121$)	82.14
OVO-SVM & PCA ($\mathcal{M} = 121$)	77.68

Table 3. Recognition Accuracy for SVM methods.

4.4. Results

The recognition of both Multi-class SVM methods outperform NN classification. Recognition accuracy is summarised at Table 4.4 for both SVM methods and both raw and compressed images. When compressed images (PCA) were used, we noticed that standardisation of the projection coefficients resulted in performance improvement, especially in the case of RBF kernels, which scored very poorly because of overfitting in the case of non-normalised features. Examples of success and failure test images of the optimal classifiers are also provided at Figure 12.

The sparsity of SVM method is evident from the small mean number of support vectors, 31 for OVR and 15 for OVO. As a result of this, mean test time is significantly faster (100 times less for a single image and 10 times less for whole testing set) than train time. This is opposed to Nearest Neighbors classifiers, whose training time is negligible (data copy), while testing time is significant. Examples of support vectors are also illustrated in Figure 13.

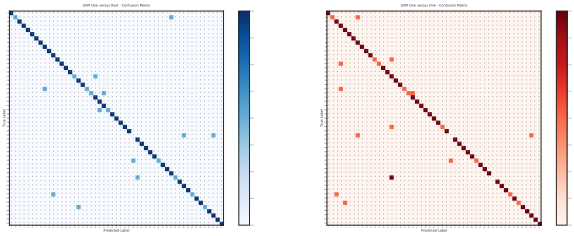

Figure 6. SVM Confusion Matrices, OVR (left) and OVO (right).

References

[1] M. Brookes. Gram Matrix, 2017. [Online]. Available: `http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/property.html`. [Accessed: 07- Dec- 2017].

[2] Paul Honeine. An eigenanalysis of data centering in machine learning, 2014. [Online]. Available: `https://arxiv.org/pdf/1407.2904.pdf`. [Accessed: 14- Dec- 2017].

[3] Z. Kolter, C. Do. Linear Algebra Review and Reference, 2015. [Online]. Available: `http://cs229.stanford.`

edu/section/cs229-linalg.pdf. [Accessed: 07- Dec-2017].

Appendix

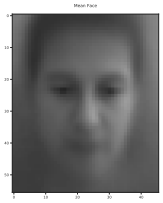

Figure 7. Mean face of the provided dataset.

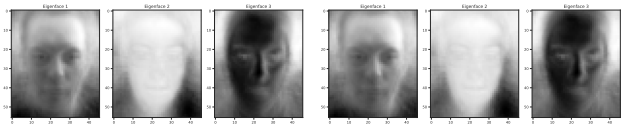

Figure 8. Three eigenfaces of $\mathcal{S}_n$ (left) and $\mathcal{S}_e$ (right).

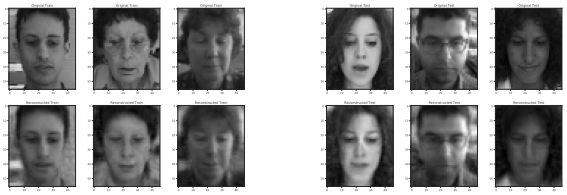

Figure 9. Comparison of original and reconstructed samples from training (left) and testing (right) data for $\mathcal{M} = 100$. Top row: original images, bottom row: reconstructed images.

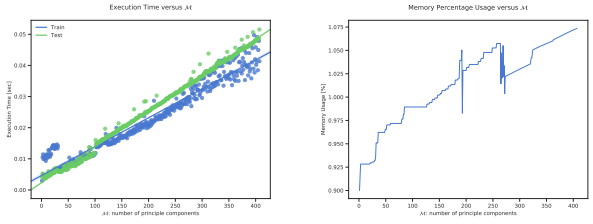

Figure 10. Nearest Neighbour Execution Time (left) and Percentage Memory Usage (right) as a function of $\mathcal{M}$. Linear complexity, $\mathcal{O}(\mathcal{M})$ time and memory complexity.

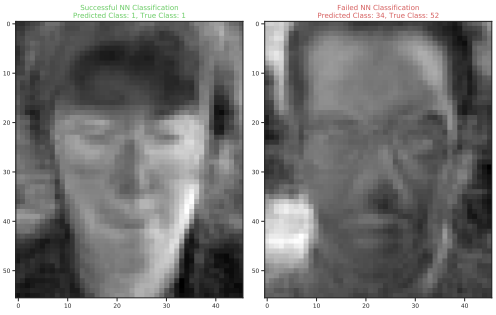

Figure 11. Nearest Neighbour Success (Left) and Failure (Right) Images examples.

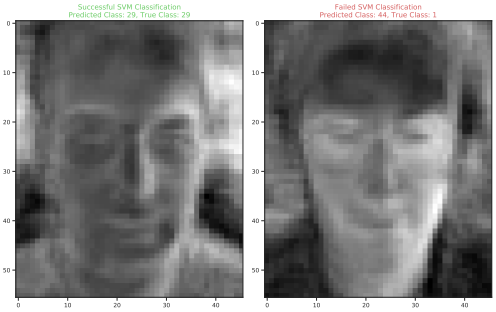

Figure 12. SVM OVR Success (Left) and Failure (Right) Images examples.

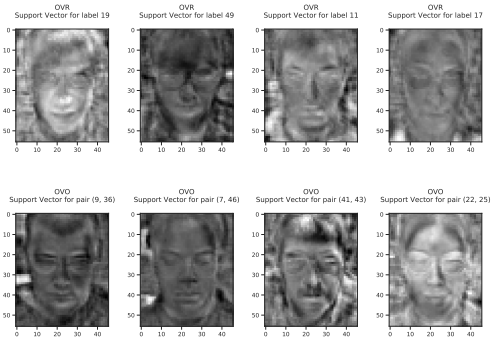

Figure 13. Support Vector Examples, OVR (top) and OVO (bottom) SVM extensions.

```python
import numpy as np
from sklearn.svm import SVC

class OVR(object):

    def __init__(self):
        # dictionary of L SVM classifiers
        self.classifiers = {}
        # list of label classes
        self.classes = []

    def fit(self, X_train, y_train):
        # L label classes
        self.classes = list(set(y_train.ravel()))
        for l in self.classes:
            # binarise labels
            l_train = -np.ones(y_train.T.shape)
            l_train[y_train.T == l] = 1
            # init and fit binary SVM
            svm = SVC(kernel=params['kernel'],
                      C=params['C'],
                      gamma=params['gamma'],
                      probability=True)
            svm.fit(X_train.T, l_train.ravel())
            # store l^{th} trained SVM
            self.classifiers[l] = svm

    def predict(self, X_test):
        # probability scores
        scores = []
        for l in self.classes:
            # get probability scores
            proba = self.classifiers[l]
                        .predict_proba(X_test.T)[:, 1]
            scores.append(proba)
        # find probability maximizers
        maxes = np.argmax(np.array(scores), axis=0)
        y_hat = np.array(
            list(map(lambda l: self.classes[l], maxes)))
            .reshape(1, -1)
        return y_hat
```

Listing 1. SVM One-Versus-Rest Python Class

```python
import numpy as np
from sklearn.svm import SVC
import itertools

class OVO(object):

    def __init__(self):
        # dictionary of L(L-1)/2 SVM classifiers
        self.classifiers = {}
        # list of label pair combinations
        self.combinations = []

    def fit(self, X_train, y_train):
        # L label classes
        classes = list(set(y_train.ravel()))
        # L(L-1)/2 pair combinations of labels
        self.combinations = list(
            itertools.combinations(classes, 2))
        # iterate over combinations
        for c1, c2 in self.combinations:
            # binarise labels
            y = np.empty(y_train.T.shape)
            y[:] = np.nan
            y[y_train.T == c1] = 1
            y[y_train.T == c2] = 0
            # indeces of interest
            _index = ~np.isnan(y).ravel()
            # select the training examples
            X = X_train.T[_index]
            # select the label examples
            y = y[_index].ravel()
            # init SVM binary classifier
            svm = SVC(kernel=params['kernel'],
                      C=params['C'],
                      gamma=params['gamma'])
            # fit SVM
            svm.fit(X, y)
            # store (c1, c2) trained SVM
            self.classifiers[(c1, c2)] = svm

    def predict(self, X_test):
        # votes leaderboard
        votes = np.zeros(
            (len(self.combinations) + 1,
            X_test.shape[1]))
        # iterate over combinations
        for c1, c2 in self.combinations:
            # get votes
            preds = self.classifiers[(c1, c2)]
                        .predict(X_test.T)

            for j, s in enumerate(preds):
                c = c1 if s == 1 else c2
                votes[c, j] += 1
        # choose the class with most votes
        y_hat = np.argmax(votes, axis=0)
        return y_hat
```

Listing 2. SVM One-Versus-One Python Class