Face recognition using a fuzzy Laplacianface classifier

Kejdi Domi

Computer Engineering Department

Epoka University, Tirana, Albania

ABSTRACT

Face recognition is the preferred method for biometric identification nowadays. This paper aims to show the advantages of using Laplacianface extraction method together with a fuzzy KNN classifier to recognize faces. We try to find the most suitable parameters for the algorithm to give the best results: this is done in a series of steps. The tests are run against the ORL database for facial recognition, and the results are compared with other papers that use the ORL database. Our method gives a satisfactory result of 98.75% accuracy, which is a great trade off considering that the algorithm takes less than 5 seconds to train.

Keywords: *Supervised learning, PCA, LDA, LPP, Laplacianface, Eigenface, Fisherface, Face recognition, Fuzzy logic, K-nearest neighbour.*

1 INTRODUCTION

Face recognition has seen an increased interest in research due to the ever-growing need for privacy and security in today's world. Face recognition is therefore closely related to biometrics. Biometrics are body measurements and calculations related to human characteristics. Face recognition is a natural intuitively appealing and straightforward biometric method.

In practice, face recognition is a very difficult problem due to a substantial variation in light direction, different face poses, and diversified facial expressions. The most well-known classification techniques used for face recognition are those of eigenface [1] and Fisherface [2]. Eigenface method tries to project the so called high dimensional face space images into a low-dimensional feature space utilizing Principal Component Analysis (PCA) [3]. There are numerous extensions to the standard PCA such as mixture-of-eigenfaces [4], topological PCA [5], kernel PCA [6], eigen-wavelet method [7], to name a few of them. In spite of these expansions, PCA still retains unwanted variations due to different conditions caused by lighting and facial expressions [2].

The Fisherface variation is insensitive to some variable conditions such as variations in illumination conditions and different facial expressions. It uses both PCA and Fisher's Linear Discriminant (FLD). It is worth stressing that by maximizing the ratio of between-scatter matrix and within-scatter matrix, FLD produces well separated classes in a low-dimensional subspace, even under severe variation in lighting and facial expressions. Also it is worth mentioning that FLD is similar to Linear Discriminant Analysis (LDA) [2], another method that is heavily utilized in face recognition problems.

The approach we are using utilizes Laplacianfaces [8]. By using locality preserving projections (LPP) [9], the face images are mapped into a face subspace for analysis. For the classification of test faces into classes we implement a fuzzy classifier. The use of a fuzzy classifier is chosen as observations have shown that a binary classifier (yes-no) is robust and much sensitive to lighting variations or facial expressions, while a fuzzy classifier quantifies the impact of such conditions allowing for flexibility in making decisions. Interestingly, the idea of such "fuzzification" of class assignment has been around for a long time and can be dated back to the results published by Keller et al. [10]

2 Literature Review

Face recognition methods are divided in 3 categories: Holistic Matching Methods, Feature-based (structural) Methods, and Hybrid Methods. [11] PCA, LDA are Holistic Matching Methods together with FLD, Laplacianface and Independent Component Analysis (ICA). This paper explores the basic methods of PCA and LDA used respectively in Eigenfaces (PCA), and Fisherfaces (PCA + LDA) as well as the newly introduced method of Laplacianface coupled with a fuzzy classification system.

2.1. Eigenface method (PCA)

PCA is a well-known technique commonly exploited in multivariate linear data analysis. The main underlying concept is to reduce the dimensionality of a data set while retaining as much variation as possible in the data set. Each image can be represented by a n × n array of pixels, the corresponding image z_i is an n^2 vector in the dimension n^2. If the training set of N faces is $Z = \{z_1, z_2, \ldots, z_N\}$ the corresponding covariance matrix is given by:

$$R = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{z}_i - \bar{\mathbf{z}})(\mathbf{z}_i - \bar{\mathbf{z}})^{\mathrm{T}} = \Phi\Phi^{\mathrm{T}}, \qquad (1)$$

where

$$\bar{\mathbf{z}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{z}_i. \qquad (2)$$

In this case z-hat represents the mean face.

Let $E = (e_1, e_2, \ldots, e_r)$ be a matrix formed by r eigenvectors corresponding to r largest eigenvalues. Thus, for a set of original face images Z, their reduced feature vectors $X = (x_1, x_2, ..., x_N)$ are obtained by projecting them into the PCA-transformed space following the linear transformation:

$$\mathbf{x}_i = E^{\mathrm{T}}(\mathbf{z}_i - \bar{\mathbf{z}}) \qquad (3)$$

With x_i being the result of this transformation.

The eigenvectors of E can be though of as a set of features that together characterize the variation between face images. Each image location contributes more or less to each eigenvector, so that we can display the eigenvector as a sort of ghostly face which we call an eigenface [1]. See Figure 1. For a face image and its eigenfaces. Each individual face can be represented in terms of linear representations of eigenfaces and can also be approximated by using the best eigenfaces (those with the largest eigenvalues) that account for the most variance in the set of face images.

Let us again stress that in virtue of the PCA calculations that do not involve any class information, the method is not capable of taking advantage of this discriminatory aspects subsequently being faced with quite a detrimental classification performance. This is shown in Figure 2.

Figure 1: A face and 12 eigenfaces derived from this face.

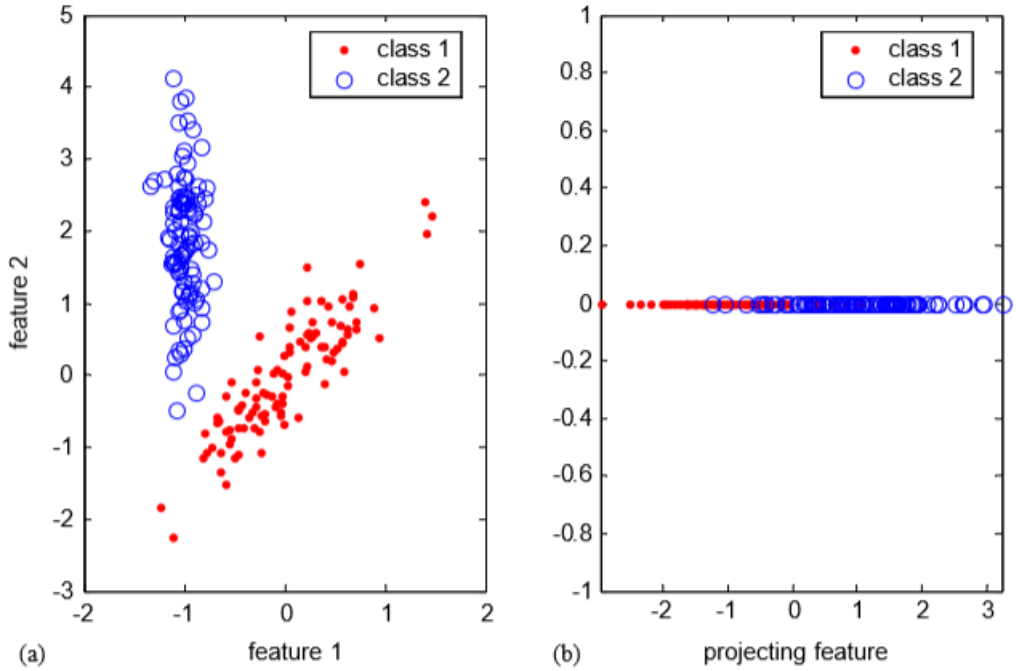

Figure 2: PCA projection for a tow-class problem: (a) original data set and (b) its reduced PCA transformation.

2.2. Fisherface Method (PCA + LDA)

The shortcomings of PCA are apparent in Figure 2. This is as a result of prioritizing the greatest variance that the data can exhibit in a lower dimensional space. In short terms, PCA does not take into consideration the within-class scatter, as it prioritizes between-class scatter. Consequently, the projected samples for each class may not be well clustered and result in the patterns being smeared together [2]. Fisherfaces algorithm introduces a new way to look at things. It defines S_B and S_W as between-class scatter matrix and within-class scatter matrix respectively.

$$S_B = \sum_{i=1}^{c} N_i (\mathbf{m}_i - \overline{\mathbf{m}})(\mathbf{m}_i - \overline{\mathbf{m}})^T, \qquad (5)$$

Where N_i is the number of vectors in the i-th class C_i and m-hat stands for the mean of all vectors (images) m_i is the mean of vectors transformed by PCA and dealing with class C_i.

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in C_i} (\mathbf{x}_k - \mathbf{m}_i)(\mathbf{x}_k - \mathbf{m}_i)^T = \sum_{i=1}^{c} S_{W_i}, \qquad (6)$$

where S_{Wi} is the covariance matrix of class C_i.

Then the optimal projection matrix W_{FLD} is chosen as:

$$W_{FLD} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|} = [\mathbf{w}_1 \quad \mathbf{w}_2 \quad \cdots \quad \mathbf{w}_m], \quad (7)$$

The same data shown in Figure 2a is now treated with FLD projection (a variation of LDA) and shown in Figure 3.

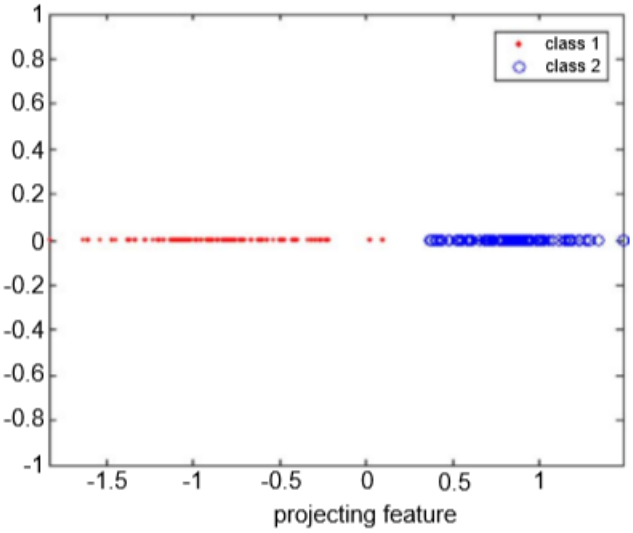

Figure 3: FLD projection for the two-class synthetic data.

2.3. Laplacianfaces method

Laplacianface method was introduced in [8]. It resembles the PCA and LDA in certain aspects but has a higher tolerance for unwanted variations resulting from lighting, facial expressions and pose by eliminating or reducing them. By using Locality Preserving Projections (LPP), the face images are mapped into a face subspace for analysis. Different from Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) which effectively see only the Euclidean structure of face space, LPP finds an embedding that preserves local information, and obtains a face subspace that best detects the essential face manifold structure [8]. The Laplacianfaces are the optimal linear approximations to the eigenfunctions of the Laplace Beltrami operator [12] on the face manifold.

In [8] the author goes in details about LPP a locality preserving algorithm. The basic points he makes I try to summarize as follows:

1. LPP seeks to preserve the intrinsic geometry of the data and local structure. The objective function of LPP is as follows:

$$\min \sum_{ij} (y_i - y_j)^2 S_{ij}, \qquad (6)$$

where y_i is the one-dimensional representation of x_i and the matrix S is a similarity matrix. The objective function with our choice of symmetric weights S_{ij} ($S_{ij} = S_{ji}$) incurs a heavy penalty if neighbouring points x_i and x_j are mapped far apart

2. This objective function can be expressed as $w^T X L X^T w$ following a series of algebraic steps, where $L = D - S$ is the Laplacian matrix $X = [x_1; x_2; ...; x_n]$, and D is a diagonal matrix; its entries are column (or row since S is symmetric) sums of S, $D_{ii} = \sum_j S_{ji}$.
3. The two matrices XLX^T and XDX^T are both symmetric and positive semidefinite since the Laplacian matrix L and the diagonal matrix D are both symmetric and positive semidefinite.
4. The Laplacian matrix for finite graph is analogous to the Laplace Beltrami operator on compact Riemannian manifolds. While the Laplace Beltrami operator for a manifold is generated by the Riemannian metric, for a graph it comes from the adjacency relation.

$$\min_{\|f\|_{L^2(M)}=1} \int_M \|\nabla f\|^2 \qquad (13)$$

which is equivalent to

$$\min_{\|f\|_{L^2(M)}=1} \int_M L(f) f, \qquad (14)$$

5. This derivation, by special graph theory [13] can lead to the following eigenvalue problem:

$$XLX^T w = \lambda XDX^T w \qquad (15)$$

For details see [13] [9] [14].

The findings of [8] are summarized in the following tables: Table 1 and Table 2. In this approach a nearest-neighbour classifier is used for classification. (We observe better results if a fuzzy classifier is used <= my claim)

TABLE 1
Performance Comparison on the Yale Database

Approach	Dims	Error Rate
Eigenfaces	33	25.3%
Fisherfaces	14	20.0%
Laplacianfaces	**28**	**11.3%**

TABLE 2
Performance Comparison on the PIE Database

Approach	Dims	Error Rate
Eigenfaces	150	20.6%
Fisherfaces	67	5.7%
Laplacianfaces	**110**	**4.6%**

2.4. Fuzzy Logic and Face Recognition

As extended in [10], a Fuzzy k-nearest classifier system aims as all classifiers the differentiation of a group of objects into N sets which we call classes. In contrast to binary methods (yes-no), a fuzzy classifier uses a range of similarity, e.g. we cannot determine with 100% certainty the weather for today thus we use sayings as precipitation 60% or sunny weather 33%. In our case, given a set of feature vectors transformed by the LPP, X = {x_1, x_2, ..., x_N}, a fuzzy "c"-class partition of these vectors specifies the degrees of membership of each vector to the classes. As usual, the partition matrix denoted by U =[μ_{ij}] for i =1, 2,...,c, and j =1,2,...,N satisfies two obvious properties

$$\sum_{i=1}^{c} \mu_{ij} = 1,$$

$$0 < \sum_{j=1}^{N} \mu_{ij} < N.$$

The computations of the membership degrees are realized through a sequence of steps:

1. Compute the Euclidean distance matrix between pairs of feature vectors in the training.
2. Set diagonal elements of this matrix to infinity (practically place large numeric values there).
3. Sort the distance matrix (treat each of its column separately) in an ascending order. Collect the class labels of the patterns located in the closest neighborhood of the pattern under consideration (as we are concerned with "k" neighbors, this returns a list of "k" integers).
4. Compute the membership grade to class "i" for jth pattern using the expression proposed in the literature [10]

$$\mu_{ij} = \begin{cases} 0.51 + 0.49(n_{ij}/k) & \text{if } i = \text{the same as the label of the } j\text{th pattern,} \\ 0.49(n_{ij}/k) & \text{if } i \neq \text{the same as the label of the } j\text{th pattern.} \end{cases}$$

Getting back to section 2.3. we can substitute the nearest-neighbour classifier with a fuzzy classifier and expect even better results.

3 Materials

3.1 Dataset

The ORL Database of Faces contains 400 images from 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). The size of each image is 92x112 pixels, with 256 grey levels per pixel. Here are some samples from the dataset:

Figure 4: ORL dataset

3.1.1 Pre-processing steps

The dataset is already set up is such a way that it does not need pre-processing. The pictures are already turned into greyscale images and they are all scaled to the same dimensions. The face is in the centre of the image (the nose is close to the centre of the image).

3.1.2 Feature extraction

Feature extraction is done through LPP (Locality Preserving Projections) [15]. Locality Preserving Projections are linear projective maps that arise by solving a variational problem that optimally preserves the neighbourhood structure of the data set. LPP may be conducted in the original space or in the reproducing kernel Hilbert space into which data points are mapped.

Figure 5 aims to show how LPP works.

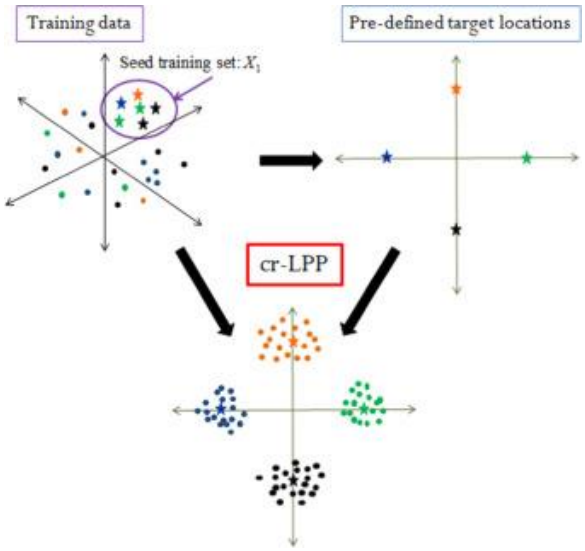

Figure 5: Locality Preserving Projections

3.2 Software Components

3.2.1. Software choice

I decided to use Java as a programming language due to its diversity and ease of use, bundled with Eclipse IDE. The version of java runtime environment I used was 1.8.0_51. Some more information can be seen in Figure 6. The project was imported from a GitHub repository (https://github.com/wihoho/FaceRecognition) and suited to our needs. I changed only the KNN class (without changing the name as it would cause problems) by turning it in a Fuzzy classification algorithm. Below in figure 7 you can see the modification of this class.

```
C:\Users\ASUS>java -version
java version "1.8.0_51"
Java(TM) SE Runtime Environment (build 1.8.0_51-b16)
Java HotSpot(TM) 64-Bit Server VM (build 25.51-b03, mixed mode)
```

Figure 6: Java version

```
static String classify(projectedTrainingMatrix[] neighbors) {
    HashMap<String, Double> map = new HashMap<String, Double>();
    int num = neighbors.length;

    for (int index = 0; index < num; index++) {
        projectedTrainingMatrix temp = neighbors[index];
        String key = temp.label;
        if (!map.containsKey(key))
            map.put(key, 1 / temp.distance);
        else {
            double value = map.get(key);
            value += 1 / temp.distance;
            map.put(key, value);
        }
    }
```

Figure 7: Fuzzy KNN classification

3.2.2. Project file composition

The project is divided in 2 main folders: faces and src folder. Bin folder is created by eclipse to store .class files and .settings folder is also accessed by eclipse to view and store settings for the project. Faces folder is a folder containing the ORL database and src folder contains 3 more folders: Jama - the folder containing utilities for matrix manipulations like CholeskyDecomposition.java or SingularValueDecomposition.java, Training – a folder containing all the necessary algorithms for classification and dimensionality reduction also some helperfiles like FileManager.java, Validation – a folder which contains PerformanceTest.java a testing unit built to give results. I added a part of project tree in the below Figure 8.

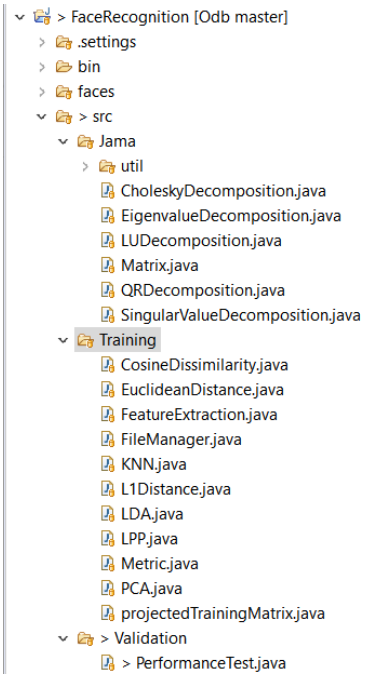

Figure 8: Part of project tree

4 Methods

The whole project can be summarized by the arguments given to a function defined in PerformanceTest.java file called test. Test takes 5 parameters: metric type, energy percentage, feature extraction mode, training number and number of clusters for Fuzzy classification. The first parameter, metric type, can be 0, 1, or 2, differentiating between cosine dissimilarity [16], L1 distance [17] and Euclidian Distance [18] respectively. During our tests we are free to choose and observe which algorithm performs better by applying different metrics to it. The second parameter, energy percentage does not apply to LLP but applies to PCA which is used by LPP and it signifies the number of dimensions or components we can choose to keep to represent a face. This is tightly connected with dimensionality reduction, thus is in direct proportion with complexity, accuracy and in indirect proportion with efficiency. The third parameter is of great importance; it allows us to choose between 3 methods implemented in this project, 0 for PCA (Eigenface space), 1 for LDA (Fisherface) and 2 for LLP (Laplacianface space); we will be using 2 in all cases as we are not concerned with the other 2 methods for now, however I added a section in the conclusions to give some perspective on the results of the other 2 algorithms. The fourth parameter, training number, is the number of photos we choose to train our database on. It must be a number from 1 to 9 as we have 10 photos for subject. It is obvious that if we increase this number, we tend to get more accurate results. The photos are chosen randomly each time and what remains is used for validation testing. The fifth parameter defines the fuzzy clusters, it is a number required by the fuzzy classifier.

What happens when we run the program can be summarized in these steps:

1. A metric is chosen
2. The call to LPP is made with all the necessary parameters
3. LPP computes PCA by calling PCA class
4. A matrix S consisting of nearest neighbours of PCA matrix calculation is computed
5. A scaling matrix D is computed as S + an indexed column
6. A matrix L is computed as D – S
7. The Laplacian operation explained above takes place
8. The feature extraction phase ends and returns to the main test file to be turned into an image
9. Validation images go through the same process and get compared to the database
10. The accuracy is computed and shown on screen

5 Results

In order to calculate results, I created a pool of different parameters with which the function *test* could be run. The third and fifth parameter were fixed; I fixed the method we are using which is Laplaceanface and the number of nearest neighbours so I could change it afterwards. I iterated through each allowed value of the parameters, running the function 135 times to get 135 different results. The results were saved together with the parameters as comma separated values in *final.csv* file. These results needed some further investigation as occasionally I would get values like those shown in figure 9 below:

```
1,30,2,8,5,0.9625
1,30,2,9,5,1.0
1,40,2,1,5,NAN
1,40,2,2,5,0.68125
```

Figure 9: 5 parameters with which I ran the test function and the result saved in a comma separated value file.

As we can see, a perfect score of 1 is shown to have been the accuracy, but this ism, possibly, far from the truth, so the policy I followed was to run the test function 4 more times with the same parameters and the 1.0 would be exchanged with the average of these results shown in figure 10. Of course, the same procedure can be followed for NAN values shown in figure 9.

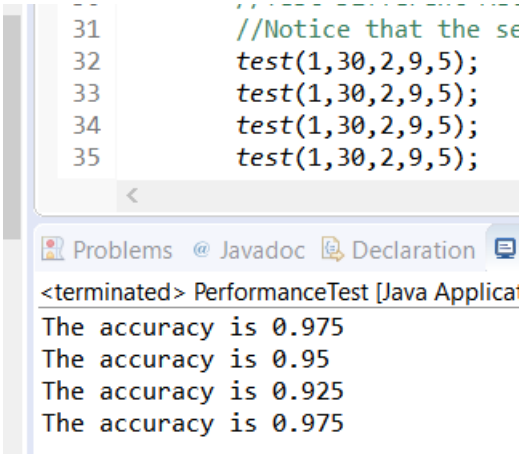

Figure 10: Test runs to "get rid" of 1.0 or NAN

Out of 135 results, only 6 were NAN and 4 gave a result of 1.0. Those with 1.0 result were substituted by the method shown above, and those that continued to give NAN result even after trying the method were substituted by the mean average of the whole dataset.

To give some perspective for what we are looking at I tried to describe the result data I gathered in table 3 below:

Table 3: Describing the newly created dataset

	metricType	energyPercentage	FeatureExtractionMode	trainNums	knn_k	accuracy
count	135.000000	135.000000	135.0	135.000000	135.0	135.000000
mean	1.000000	30.000000	2.0	5.000000	5.0	0.843775
std	0.819538	14.194807	0.0	2.591605	0.0	0.113069
min	0.000000	10.000000	2.0	1.000000	5.0	0.486111
25%	0.000000	20.000000	2.0	3.000000	5.0	0.788393
50%	1.000000	30.000000	2.0	5.000000	5.0	0.875000
75%	2.000000	40.000000	2.0	7.000000	5.0	0.925000
max	2.000000	50.000000	2.0	9.000000	5.0	0.987500

As we can see, the maximum accuracy is **0.9875**. This accuracy came from running the *test* function with parameters 1,40,2,8,5, in other words, we chose L1 distance to compare an image generated by 40 selected features using LPP. Out of 10 images from the database for each subject, 8 were used for training and 2 for testing. We also used a fuzzy KNN classifier with k = 5 nearest neighbours. The

average accuracy of our algorithm was also satisfactory, 0.8438, however with 0.1131 standard deviation this result must be viewed carefully as a mere representation of how certain parameter decisions affect our algorithm. Lowest accuracy that I got on this test run was 0.4861, attributed to a test function run with these parameters: 1,30,2,1,5. This minimum accuracy is the result of low training samples (here only 1 image per person is used for training). In further examinations we must keep a fixed number of 8 training samples to show how algorithm performs in other conditions.

The previous statement makes an assumption that choosing 8 training samples is the best practice, and this assumption is confirmed when we have a look at trainNums against accuracy in figure 11:

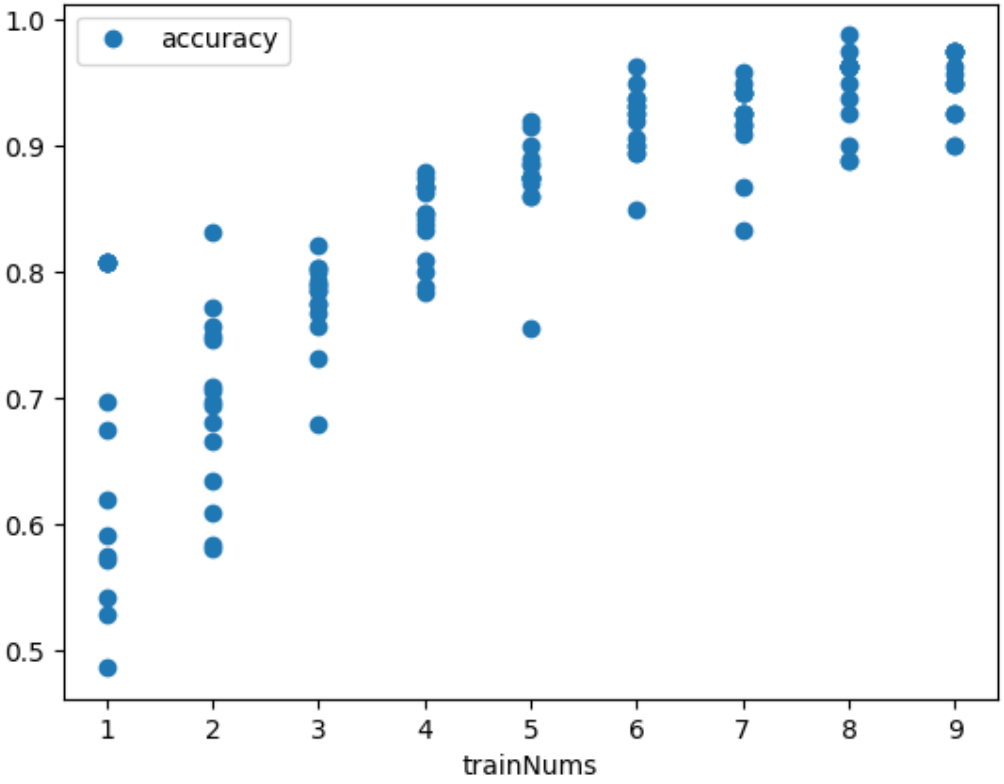

Figure 11: Training samples against accuracy

From this data we can say that training number is important to accuracy. The contrary, however, is true for energy percentage or number of features as we call it. Figures 12 and 13 show clearly that this variable has a weak correlation with accuracy.

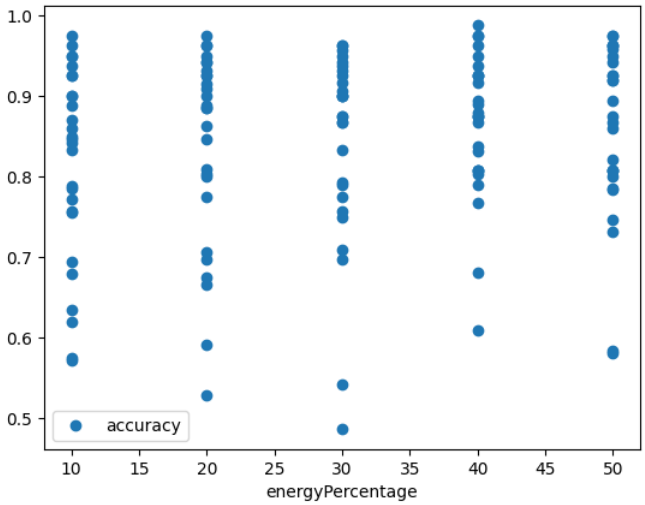

Figure 12: Number of features against accuracy

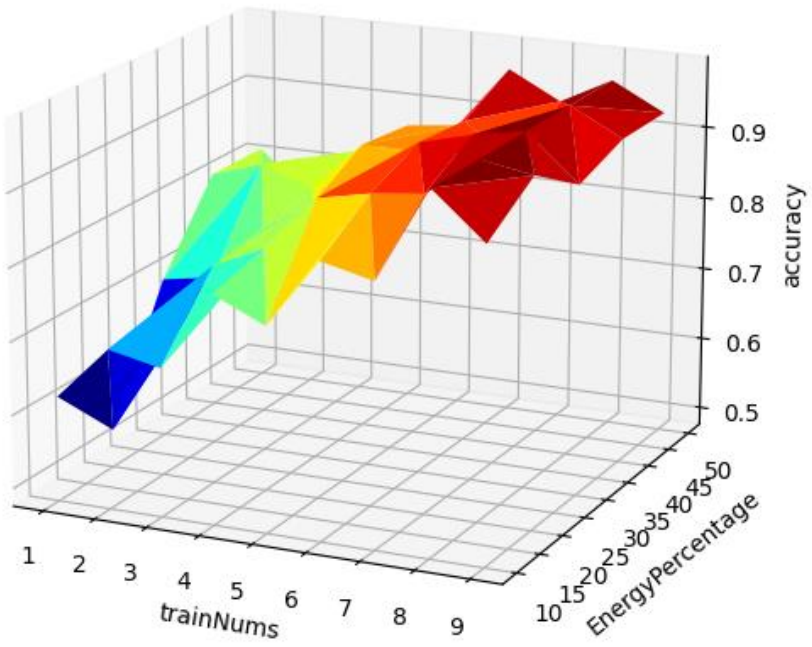

Figure 13: 3D plot of trainNums, energyPercentage and accuracy

This could be due to the narrow range of energy percentage (from 10 to 50), so we could investigate this variable if we keep all other variables fixed.

So far, the data has shown that the highest accuracy is achieved using 8 test samples, L1 distance metric, and we agree not to change number of features for now. Let us take a look at how the accuracy changes if we choose the number of neighbours from the range (5, 10…, 100). Figure 14 shows raw data and figure 15 shows 2 regression lines, linear regression in green and quadratic regression in orange. As we can see from figure 15, quadratic regression proves to be more efficient; as a result, we can conclude that k=5 would be the best alternative for further investigation.

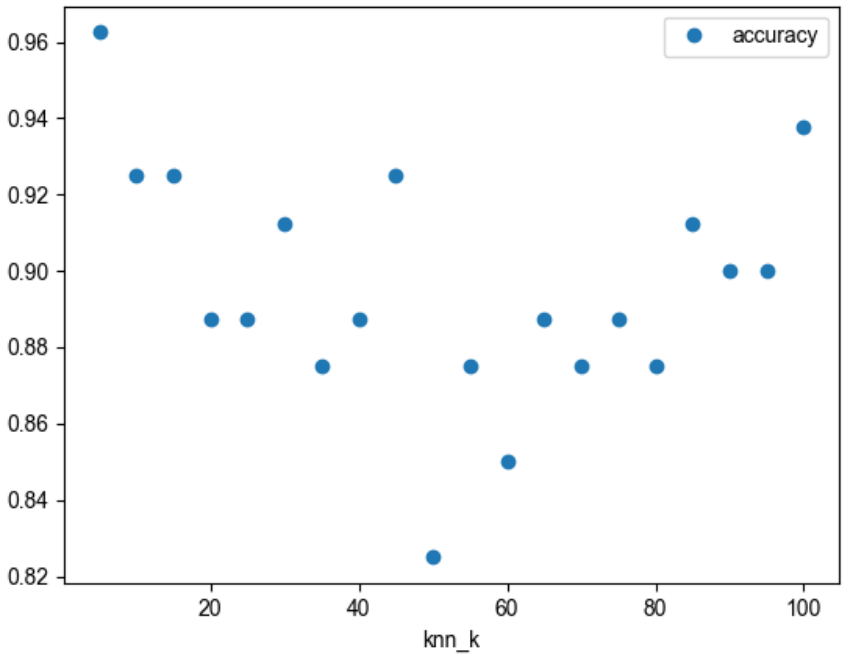

Figure 14: Accuracy affected by number of nearest neighbours

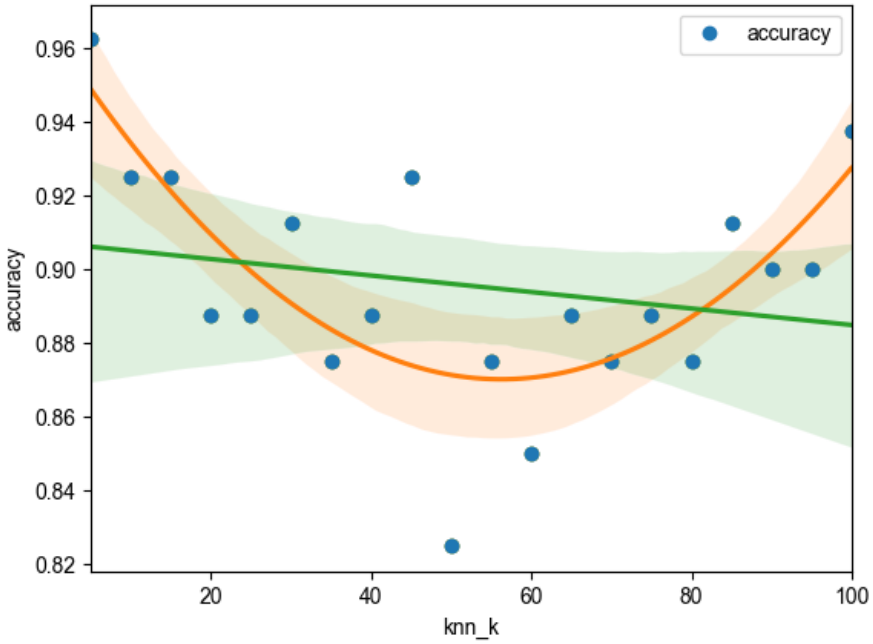

Figure 15: Regression models on the previous figure

It is worth noting that the lowest accuracy we got was 0.825, and this proves once again that choosing parameters 1,40,2,8,5 was the best choice. Now, at last, we can examine the number of features and how it really affects the accuracy. We choose from the range (10, 20, …, 100) and try to come to a conclusion to how the number of features affects our accuracy. Figure 16 and 17 help us deduce that the general trend for accuracy is to go down as more features are selected.

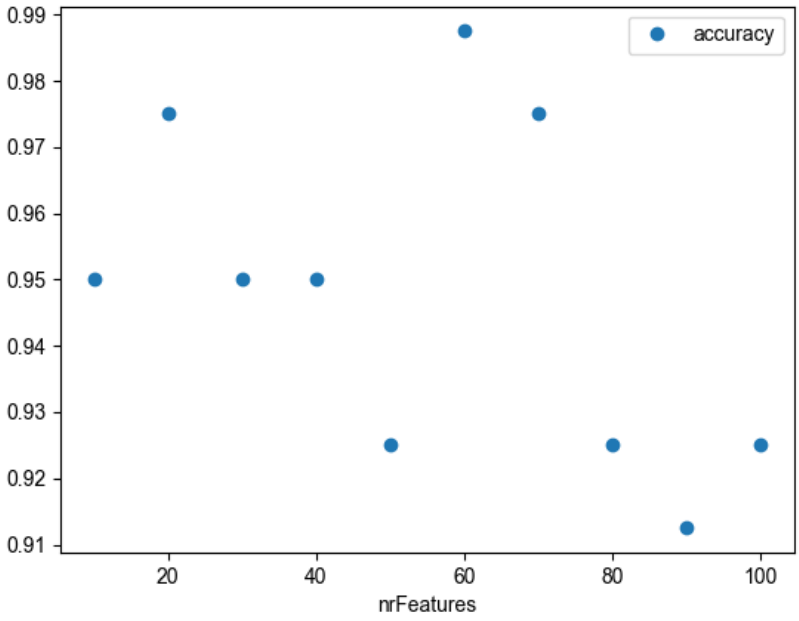

Figure 16: Raw data between number of features and accuracy

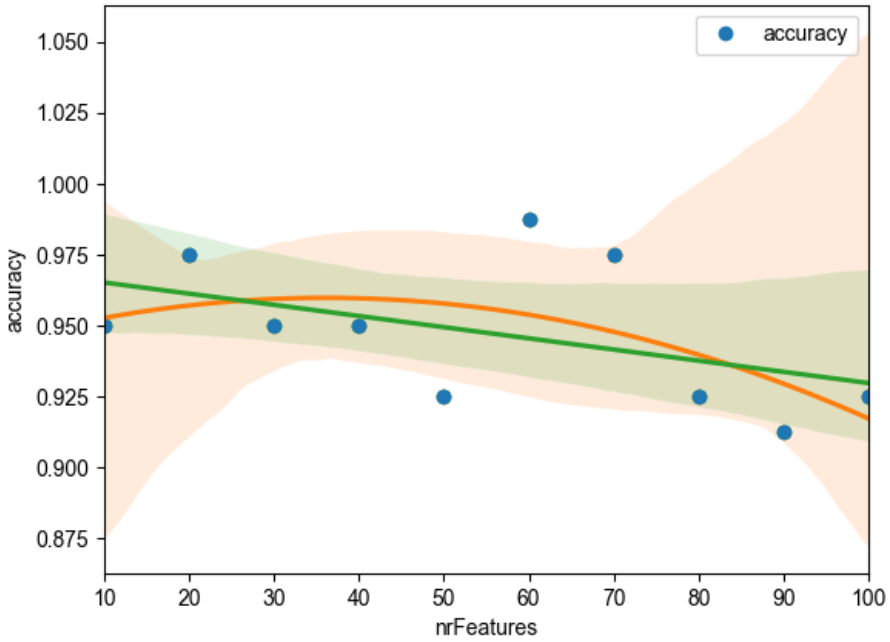

Figure 17: Regression models (orange => order = 2, green => order = 1) for the previous data.

Also, it is worth noting that, although, 60 features and 70 features give the best results, the standard deviation is small and each point gives a good enough accuracy.

6 Conclusion

In conclusion, Laplacianface algorithm for face recognition, combined with fuzzy KNN classification gives a satisfying result of 98.75% accuracy. The algorithms were implemented in java, while the data was analysed and visualized using Python and CSV files. This implementation of facial recognition was done using the ORL face database. To show the effectivity of my method, and to compare it with other methods, I researched some papers written using ORL database and summarized my findings in table 4.

Table 4: Comparison of different methods.

Ref	Method	Percentage of correct classification (PCC)	Notes
[19]	Hybrid NN: SOM+a convolution NN	96.2%	DB contained 400 images of 40 individuals. The classification time less than .5 second for recognizing one facial image, but training time is 4 hours.
[20]	Hidden Markov model (HMMs)	87%	
[20]	A pseudo 2DHMM	95%	Its classification time and training time were not given (believe to be very expensive)
[21]	SVM with a binary tree	91.21% for SVM and 84.86% for Nearest Centre Classification (NCC)	They compare the SVMs with standard eigenface approach using the NCC
[22]	Optimal-Pairwise coupling (O-PWC) SVM	PWC achieved 95.13% , O-PWC (cross entropy) achieved 96.79% and OPWC (square error) achieved 98.11%	They select 200 samples (5 for each individual) randomly as training set. The remaining 200 samples are used as the test set.
[23]	Several SVM+NN arbitrator	97.9%	An average processing time of .22 second for face pattern with 40 classes. On the same DB, PCC for eigenfaces is 90% and for pseudo-2D HMM is 95% and for convolutional NN is 96.2%
[1]	Eigenface	90%	
[24]	PDBNN	96%	PDBNN face recognizing up to 200 people in approximately 1 second and the training time is 20 minutes
[25]	MRF	86.95 (when using 5 images for training and 6 for testing)	
	Laplacianface + Fuzzy KNN classification	**98.75%**	**I test the recognition accuracy with different number of training samples to find out 8 training samples give the best result. Result is shown in less than 5 seconds.**
[26]	A combined classifier uses the generalization capabilities of both Learning Vector Quantization (LVQ) and Radial Basis Function (RBF) neural networks to build a representative model of a face from a variety of training patterns with different poses, details and facial expressions	99.5%	A new face synthesis method is implemented for reducing the false acceptance rate and enhancing the rejection capability of the classifier. The system is capable of recognizing a face in less than one second.

[27]	Build face recognition committee machine (FRCM) of Eigenface, Fisherface, Elastic Graph Matching (EGM), SVM, and Neural network	FRCM gives 98.8% and it outperforms all the individual on average	They adopt leaving-one-out cross validation method.
[28]	Boosted parameter-based combined classifier	100%	The DB is divided into 200 images (10 for each person) for training and 200 for testing (10 for each person)

References

[1] M. T. a. A. Pentland, "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience,* vol. 3, no. 1, pp. 71-86, 1991.

[2] J. H. D. K. P.N. Belhumeur, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 19, no. 7, pp. 711-720, 1997.

[3] L. S. a. M. Kirby, "Low-dimensional procedure for the characterization of human faces," *Journal of the Optical Society of America A,* vol. 4, no. 3, p. 519, 1987.

[4] D. K. S. B. H.C. Kim, "Face recognition using the mixture-of-eigenfaces method," *Pattern Recognition Lett.,* vol. 23, p. 1549–1558, 2002.

[5] J. V. F. L. J. V. A. Pujol, "Topological principal component analysis for face encoding and recognition," *Pattern Recognition Lett.,* vol. 22, p. 769–776, 2001.

[6] K. J. H. K. K.I. Kim, "Face recognition using kernel principal component analysis," *IEEE Signal Process. Lett.,* vol. 9, no. 2, pp. 40-42, 2002.

[7] Y. L. B. Li, "When eigenfaces are combined with wavelets," *Knowledge Systems,* vol. 15, p. 343–347, 2002.

[8] S. Y. Y. H. P. N. a. H.-J. Z. Xiaofei He, "Face recognition using Laplacianfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 27, no. 3, pp. 328-340, 2005.

[9] X. a. P. N. He, "Locality preserving projections.," *Advances in neural information processing systems,* vol. 16, no. 16, pp. 153-160, 2004.

[10] M. G. J. G. J.M. Keller, "A fuzzy k-nearest neighbor algorithm," *IEEE Trans. Syst. Man Cybernet,* vol. 15, no. 4, p. 580–585, 1985.

[11] B. B. M. Divyarajsinh N. Parmar, "Face Recognition Methods & Applications," *International Journal of Computer Technology & Applications,* vol. 4, no. 1, pp. 84-86, 2013.

[12] H. Flanders, Differential forms with applications to the physical sciences, Dover, 1989.

[13] F. Chung, "Spectral Graph Theory," *Proc. Regional Conf. Series in Math.,,* vol. 92, 1997.

[14] M. B. a. P. Niyogi, "Laplacian Eigenmaps and Spectral Techniques for Embedding and

Clustering,," *Proc. Conf. Advances in Neural Information Processing System,* vol. 15, 2001.

[15] X. &. N. P. He, "Locality preserving projections. Advances in neural information processing systems," vol. 16, no. 16, pp. 153-160, 2004.

[16] H. V. a. L. B. Nguyen, "Cosine similarity metric learning for face verification.," *Asian conference on computer vision. Springer,* pp. 709-720, 2012.

[17] J. L. Lagrange, Méchanique analitique, Vve Desaint, 1788.

[18] T. L. H. a. D. D. Euclid, Euclid's Elements: all thirteen books complete in one volume : the Thomas L. Heath translation, Santa Fe, N.M: Green Lion Press., 2002.

[19] C. G. A. T. a. A. B. S. Lawrence, "Face recognition: A convolutional neural-network approach," *IEEE Trans. Neural Networks,* vol. 8, pp. 98-113, 1997.

[20] F. S. a. A. Harter, ""Parameterisation of a stochastic model for human face identification"," *Proc. Second IEEE Workshop Applications of Computer Vision,* 1994.

[21] J. K. Y. P. L. a. J. M. K. Jonsson, "Support vector machines for face authentication," *British Machine Vision Conference,* p. 543–553, 1999.

[22] H. Z. S. L. G.D. Guo, "Pairwise face recognition," *Proceedings of 8th IEEE International Conference on Computer Vision.,* 9-12 July 2001.

[23] K. J. a. J. K. K. I. Kim, "Face recognition using support vector machines with local correlation kernels," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 16, no. 1, pp. 97-111, 2002.

[24] S. K. a. L. L. S.H. Lin, "Face recognition/detection by probabilistic decision-based neural network," *IEEE Trans. Neural Networks,* vol. 8, pp. 114-132, 1991.

[25] V. P. a. D. N. M. Rui Huang, "A hybrid face recognition method using Markov random fields," *ICPR (3) ,* pp. 157-160, 2004.

[26] a. A. A.-R. A.S. Tolba, "Combined classifiers for invariant face recognition," *Pattern Anal. Appl,* vol. 3, no. 4, pp. 289-302, 2000.

[27] M. L. a. I. K. Ho-Man Tang, "Face recognition committee machine," *In Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003),* pp. 837-840, 2003.

[28] A. E.-B. a. A. E.-H. A.S. Tolba, "A robust boosted parameter- based combined classifier for pattern recognition," *submitted for publication.*

APPENDIX

Filename = final.csv, description = data generated to evaluate results.

metricType,energyPercentage,FeatureExtractionMode,trainNums,knn_k,accuracy

0,10,2,1,5,0.6194444444444445

0,10,2,2,5,0.634375

0,10,2,3,5,0.6785714285714286

0,10,2,4,5,0.7875

0,10,2,5,5,0.755

0,10,2,6,5,0.85

0,10,2,7,5,0.8333333333333334

0,10,2,8,5,0.8875

0,10,2,9,5,0.9

0,20,2,1,5,0.675

0,20,2,2,5,0.696875

0,20,2,3,5,0.775

0,20,2,4,5,0.8083333333333333

0,20,2,5,5,0.885

0,20,2,6,5,0.9

0,20,2,7,5,0.9083333333333333

0,20,2,8,5,0.8875

0,20,2,9,5,0.925

0,30,2,1,5,0.6972222222222222

0,30,2,2,5,0.75

0,30,2,3,5,0.775

0,30,2,4,5,0.8666666666666667

0,30,2,5,5,0.875

0,30,2,6,5,0.93125

0,30,2,7,5,0.9166666666666666

0,30,2,8,5,0.9

0,30,2,9,5,0.9

0,40,2,1,5,0.8078697824808933

0,40,2,2,5,0.83125

0,40,2,3,5,0.7892857142857143

0,40,2,4,5,0.875

0,40,2,5,5,0.875

0,40,2,6,5,0.925

0,40,2,7,5,0.8666666666666667

0,40,2,8,5,0.9625

0,40,2,9,5,0.925

0,50,2,1,5,0.8078697824808933

0,50,2,2,5,0.746875

0,50,2,3,5,0.8214285714285714

0,50,2,4,5,0.8666666666666667

0,50,2,5,5,0.875

0,50,2,6,5,0.9625

0,50,2,7,5,0.925

0,50,2,8,5,0.9625

0,50,2,9,5,0.925

1,10,2,1,5,0.5722222222222222

1,10,2,2,5,0.69375

1,10,2,3,5,0.7857142857142857

1,10,2,4,5,0.8416666666666667

1,10,2,5,5,0.87

1,10,2,6,5,0.9

1,10,2,7,5,0.925

1,10,2,8,5,0.9625

1,10,2,9,5,0.95

1,20,2,1,5,0.5277777777777778

1,20,2,2,5,0.665625

1,20,2,3,5,0.8035714285714286

1,20,2,4,5,0.8458333333333333

1,20,2,5,5,0.915

1,20,2,6,5,0.925

1,20,2,7,5,0.9416666666666667

1,20,2,8,5,0.9625

1,20,2,9,5,0.95

1,30,2,1,5,0.4861111111111111

1,30,2,2,5,0.709375

1,30,2,3,5,0.7892857142857143

1,30,2,4,5,0.8333333333333334

1,30,2,5,5,0.875

1,30,2,6,5,0.90625

1,30,2,7,5,0.95

1,30,2,8,5,0.9625

1,30,2,9,5,0.95625

1,40,2,1,5,0.8078697824808933

1,40,2,2,5,0.68125

1,40,2,3,5,0.8035714285714286

1,40,2,4,5,0.8791666666666667

1,40,2,5,5,0.875

1,40,2,6,5,0.9375

1,40,2,7,5,0.9166666666666666

1,40,2,8,5,0.9875

1,40,2,9,5,0.975

1,50,2,1,5,0.8078697824808933

1,50,2,2,5,0.58125

1,50,2,3,5,0.7321428571428571

1,50,2,4,5,0.7833333333333333

1,50,2,5,5,0.86

1,50,2,6,5,0.91875

1,50,2,7,5,0.9416666666666667

1,50,2,8,5,0.975

1,50,2,9,5,0.95

2,10,2,1,5,0.575

2,10,2,2,5,0.771875

2,10,2,3,5,0.7571428571428571

2,10,2,4,5,0.8458333333333333

2,10,2,5,5,0.86

2,10,2,6,5,0.95

2,10,2,7,5,0.925

2,10,2,8,5,0.9375

2,10,2,9,5,0.975

2,20,2,1,5,0.5916666666666667

2,20,2,2,5,0.70625

2,20,2,3,5,0.8

2,20,2,4,5,0.8625

2,20,2,5,5,0.885

2,20,2,6,5,0.93125

2,20,2,7,5,0.9416666666666667

2,20,2,8,5,0.9625

2,20,2,9,5,0.975

2,30,2,1,5,0.5416666666666666

2,30,2,2,5,0.75625

2,30,2,3,5,0.7928571428571428

2,30,2,4,5,0.8666666666666667

2,30,2,5,5,0.9

2,30,2,6,5,0.9375

2,30,2,7,5,0.9416666666666667

2,30,2,8,5,0.925

2,30,2,9,5,0.9625

2,40,2,1,5,0.8078697824808933

2,40,2,2,5,0.609375

2,40,2,3,5,0.7678571428571429

2,40,2,4,5,0.8375

2,40,2,5,5,0.89

2,40,2,6,5,0.89375

2,40,2,7,5,0.925

2,40,2,8,5,0.95

2,40,2,9,5,0.975

2,50,2,1,5,0.8078697824808933

2,50,2,2,5,0.584375

2,50,2,3,5,0.7857142857142857

2,50,2,4,5,0.8

2,50,2,5,5,0.92

2,50,2,6,5,0.89375

2,50,2,7,5,0.9583333333333334

2,50,2,8,5,0.9625

2,50,2,9,5,0.975

Filename = knn_k_acc.csv, description = data generated for finding accuracy and k relation.

knn_k,accuracy

5,0.9625

10,0.925

15,0.925

20,0.8875

25,0.8875

30,0.9125

35,0.875

40,0.8875

45,0.925

50,0.825

55,0.875

60,0.85

65,0.8875

70,0.875

75,0.8875

80,0.875

85,0.9125

90,0.9

95,0.9

100,0.9375

Filename = nF_acc.csv, description = data generated for finding accuracy and number of features relation.

nrFeatures,accuracy

10,0.95

20,0.975

30,0.95

40,0.95

50,0.925

60,0.9875

70,0.975

80,0.925

90,0.9125

100,0.9250

Filename = results_visualized.py, description = python script to visualize different relations in the final.csv data as well as plotting relationships.

```
from numpy import double
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
data = pd.read_csv("final.csv")
# Preview the first 5 lines of the loaded data
print(data.head())
data.plot(x='trainNums', y='accuracy', style='o')
fig = plt.figure()
ax = Axes3D(fig)
ax.set_xlabel('trainNums')
ax.set_ylabel('EnergyPercentage')
ax.set_zlabel('accuracy')
ax.plot_trisurf(data.trainNums, data.energyPercentage, data.accuracy, cmap=cm.jet, linewidth=0.2)
```

```
data.plot(x='energyPercentage', y='accuracy', style='o')
plt.show()
```

Filename = find_propper_k.py, description = python code to interpret the data shown in file knn_n_acc.csv.

```
from numpy import double
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import seaborn as sns
data = pd.read_csv("knn_k_acc.csv")
# Preview the first 5 lines of the loaded data
print(data.head())
data.plot(x='knn_k', y='accuracy', style='o')
data.plot(x='knn_k', y='accuracy', style='o')
sns.set_theme(color_codes=True)
sns.regplot(x="knn_k", y="accuracy", data=data, order=2)
sns.regplot(x="knn_k", y="accuracy", data=data, order=1)
plt.show()
```

Filename = find_propper_nF.py, description = python script to find and show the relationship between number of features and accuracy.

```
from numpy import double
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import seaborn as sns
data = pd.read_csv("nF_acc.csv")
# Preview the first 5 lines of the loaded data
print(data.head())
data.plot(x='nrFeatures', y='accuracy', style='o')
```

```
data.plot(x='nrFeatures', y='accuracy', style='o')
sns.set_theme(color_codes=True)
sns.regplot(x="nrFeatures", y="accuracy", data=data, order=2)
sns.regplot(x="nrFeatures", y="accuracy", data=data, order=1)
# sns.regplot(x="nrFeatures", y="accuracy", data=data, order=3)
plt.show()
```