

RESEARCH & PROJECT SUBMISSIONS

**Program:**

***Course Code: CSE 366***

***Course Name: Pattern Recognition***

***Examination Committee***

**Dr. Mahmoud Khalil**

**Ain Shams University**

**Faculty of Engineering**

**Spring Semester – 2020**

**Student Personal Information**

|  |  |
| --- | --- |
| **Student Name:** | **Bassil Usama Mohamed Abo El Nasr El Oraby** |
| **Student Code:** | **16P6025** |
| **Class/Year:** | **Spring 2020** |

**Plagiarism Statement**

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

**Signature/Student Name:**

**16/6/2020**

**Date:**

**Bassil Usama Mohamed**

**Submission Contents**

**Part 1: Assignment Description & Analysis**

**Part 2: Project Description & Analysis**

***Part 1: Ass. Description & Analysis***

**1.1) Introduction:**

In this Report, we’ll be discussing the Assignment to which I was assigned where it was required of me to implement the Classifiers that we’ve been taught in this course’s curriculum, we’ll Describe the problem which we’re facing, the stats on the dataset I acquired and how we implemented each classifier to solve the problem.

**1.2) Problem Statement:**

We have a dataset with the following stats:

\_\_Number of Features: \_\_ 30 numeric, predictive attributes and the class.

\_\_Features Type: \_\_ Integer

\_\_Examples Number: \_\_ 569

\*\*Dataset Description: \*\*

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

\*\*Attribute information: \*\*

\* radius (mean of distances from center to points on the perimeter)

\* texture (standard deviation of gray-scale values)

\* perimeter

\* area

\* smoothness (local variation in radius lengths)

\* compactness (perimeter^2 / area - 1.0)

\* concavity (severity of concave portions of the contour)

\* concave points (number of concave portions of the contour)

\* symmetry

\* fractal dimension (“coastline approximation” - 1)

The mean, standard error, and “worst” or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

**1.3) Required Classifiers:**

1. Naïve Bayes Classifier
2. Linear Discriminant Analysis
3. Decision Trees
4. Perceptron
5. Support Vector Machines
6. Neural Network
7. Radial Basis Function

**1.3.1) Naïve Bayes Classifier:**

Naive Bayes classifiers are linear classifiers that are known for being simple yet very efficient. The probabilistic model of naive Bayesclassifiers is based on Bayes' theorem, and the adjective naive comes from the assumption that the features in a dataset are mutually independent, So I used it as a first option to find the best classification results I can get as it’s an optimal classifier, results were pretty good actually, getting an accuracy result of 97.4% and a whooping precision score of a full 100%, Code and full output is available in the ipython notebook file.

A good insight while working was that, More training points leads to more accurate results (Without overfitting the model), so for example, if I had my train\_test\_split function split my dataset into 70% training data points against 30% for the testing datapoints, lower results would be achieved rather than if I get an 80-20 split for training against testing datapoints, so the more datapoints, the better results.

**1.3.2) Linear Discriminant Analysis:**

Linear discriminant analysis (LDA), normal discriminant analysis (NDA), or discriminant function analysis is a generalization of Fisher's linear discriminant, a method used in statistics, pattern recognition, and machine learning to find a linear combination of features that characterizes or separates two or more classes, Used in the assignment, it got a fairly good result when discriminating the best vectors for classifying different classes solely and decreasing the dimensionality of the dataset, randomForest Classifier was used to asses the classification results after the reduction technique was used, and I got a result of 94% for accuracy and 93% for precision, Code and full output is available in the ipython notebook file.

**1.3.3) Decision Trees:**

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules.

In this assignment, I used the Decision tree classifier to classify either the datapoint was benign or malignant based on the questions presented by the classifier in it’s internal nodes split into a binary answer of either “yes” or “no”, some insights about Decision Tree Classifiers are that Tuning the parameters such as min\_samples\_split led to decrease in the accuracy scores, meaning that a default of value equal 2 yields better results for this dataset using this classifier, also accuracy scores when using training dataset as a ground truth matrix should yield a result of 1.0 or 100% true prediction, yet when using the minimum number of samples required to split an internal node or 'min\_samples\_split' parameter for simplicity, we observe that training dataset does not yield 1.0 result for accuracy scores and that is because increasing the value of the min\_sample\_split smoothens the decision boundary and thus prevents it from overfitting.

**1.3.4) Perceptron Algorithm:**

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

This classifier is good, scoring a result of 97.4% for accuracy and a result of 97.2% for the algorithm’s precision.

Some good insights about this algorithm are:

* Of course, it does not offer the same results as the naive Bayes classifier as it is not optimal regarding to classification activity
* The more the iterations, the less the accuracy of the predicted result, **Unless agreed upon a validation error at which when reached (Minimum cost) the algorithm halts**, yet we cannot decrease the iterations beyond a certain limit or the hidden layer will reach its maximum iterations before convergence.

**1.3.5) Support Vector Machine:**

Support Vector Machines uses the concept of**‘Support Vectors‘**, which are the closest points to the hyperplane.

In the below example, the red line denotes our decision boundary that separates the 2 classes (Blue stars and Red circles) and the hyphened lines represent our ‘**Margin’**, the gap we want between the Support Vectors of both the classes.

A close up of a map

Description automatically generated

In the Assignment, SVM scored a result of 95.6% for accuracy & a 94.6% for precision.

Some good insights regarding the SVM Algorithm:

* **Kernel**: The main function of the kernel is to transform the given dataset input data into the required form. There are various types of functions such as linear, polynomial, and radial basis function (RBF). Polynomial and RBF are useful for non-linear hyperplane. Polynomial and RBF kernels compute the separation line in the higher dimension. In some of the applications, it is suggested to use a more complex kernel to separate the classes that are curved or nonlinear. This transformation can lead to more accurate classifiers.
* **Regularization**: Regularization parameter in python's Scikit-learn C parameter used to maintain regularization. Here C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimization how much error is bearable. This is how you can control the trade-off between decision boundary and misclassification term. A smaller value of C creates a small-margin hyperplane and a larger value of C creates a larger-margin hyperplane.
* **Gamma**: A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes over-fitting. In other words, you can say a low value of gamma considers only nearby points in calculating the separation line, while the a value of gamma considers all the data points in the calculation of the separation line.

**Advantages:**

SVM Classifiers offer good accuracy and perform faster prediction compared to Naïve Bayes algorithm. They also use less memory because they use a subset of training points in the decision phase. SVM works well with a clear margin of separation and with high dimensional space.

**Disadvantages:**

SVM is not suitable for large datasets because of its high training time and it also takes more time in training compared to Naïve Bayes. It works poorly with overlapping classes and is also sensitive to the type of kernel used.

**1.3.6) Neural Network (MLP):**

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates, Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

**A Distinction between MLP & SLP that ought to be discussed:**

* **MLP:**
* Weight decay:  
  ➢Avoid overfitting by imposing the condition that weights must be small  
  ➢After each update, weights are decayed by some factor  
  ➢Related to regularization (also used in SVM)
* Number of hidden layers:  
  ➢More layers -> more complex  
  ➢Networks with more hidden layers are more prone to get caught in local minima, hence prone to classification errors  
  ➢Smaller the better (KISS)
* Overfitting is not only about the number of degrees of freedom compared to the amount of data. With flexible models like neural networks it is perfectly possible to overfit regardless of the size of the data (even if this would be huge); so it is not as simple.The main mechanism to prevent overfitting is through regularization, which aims at keeping model complexity low while fitting the data as best as possible. The optimal amount of regularization is a trade-off between model simplicity and how well it fits the data. In most learning approaches, including neural networks, this is controlled via a hyperparameter that you must set (or optimize).
* **SLP:**
* A linear classifier with a single output neuron or perceptron that does not have the same advantages as the multi layer perceptron model, but not always are the multi layers of the MLP Considered as an advantage, in this problem, the one layer perceptron or the basic perceptron model works on finding weights for the equation of the line or decision boundary that will classify both classes correctly being either benign or malignant, through adjusting the weights by a series of steps similar to the gradient descent method, reaching the best minimum cost which ideally would be zero if the two classes were linearly separable.

**1.3.7) Radial Basis Function:**

Radial Basis is a function or a kernel more than a whole classifier, meaning that its mathematical model is being used within other wholesome classifiers such as Multi-layer neural networks and Support Vector Machines, In the assignment and project I preferred using The SVM Classifier as a container for applying the RBF kernel, Actually RBF did score better results when it comes to recall with a full 100% while the linear SVM Scored 98.6% but in accuracy linear kernel did top over the RBF ones, getting an accuracy result of 95.6% while the RBF getting 94.7%, although the gap is not that wide but I thought the RBF would perform better since it’s non linear and can get more accustomed to the data and to classifying them.

***Part 2: Project Description & Analysis***

# Introduction:

Face recognition is the problem of identifying and verifying people in a photograph by their face.

It is a task that is trivially performed by humans, even under varying light and when faces are changed by age or obstructed with accessories and facial hair. Nevertheless, it is remained a challenging computer vision problem for decades until recently.

Pattern recognition methods are able to leverage very large datasets of faces and learn rich and compact representations of faces, allowing modern models to first perform as-well and later to outperform the face recognition capabilities of humans.

The 2001 paper titled “Face Detection: A Survey” provides a taxonomy of face detection methods that can be broadly divided into two main groups:

* Feature-Based.
* Image-Based.

The feature-based face detection uses hand-crafted filters that search for and locate faces in photographs based on a deep knowledge of the domain. They can be very fast and very effective when the filters match, although they can fail dramatically when they don’t, e.g. making them somewhat fragile.

Alternately, image-based face detection is holistic and learns how to automatically locate and extract faces from the entire image. Neural networks fit into this class of methods.

"**… address face detection as a general recognition problem. Image-based representations of faces, for example in 2D intensity arrays, are directly classified into a face group using training algorithms without feature derivation and analysis. […] these relatively new techniques incorporate face knowledge implicitly into the system through mapping and training schemes.**"  
— Face Detection: A Survey, 2001.

Perhaps one of the more widely known and adopted “machine learning” methods for face recognition was described in the 1991 paper titled “Face Recognition Using Eigenfaces.” Their method, called simply “Eigenfaces,” was a milestone as it achieved impressive results and demonstrated the capability of simple holistic approaches.

"**Face images are projected onto a feature space (“face space”) that best encodes the variation among known face images. The face space is defined by the “eigenfaces”, which are the eigenvectors of the set of faces; they do not necessarily correspond to isolated features such as eyes, ears, and noses**"  
— Face Recognition Using Eigenfaces, 1991.

The 2018 paper titled “Deep Face Recognition: A Survey,” provides a helpful summary of the state of face recognition research over the last nearly 30 years, highlighting the broad trend from holistic learning methods (such as Eigenfaces), to local handcrafted feature detection, to shallow learning methods, to finally deep learning methods that are currently state of the art.

"**The holistic approaches dominated the face recognition community in the 1990s. In the early 2000s, handcrafted local descriptors became popular, and the local feature learning approach were introduced in the late 2000s. […] [shallow learning method] performance steadily improves from around 60% to above 90%, while deep learning boosts the performance to 99.80% in just three years.**"  
— Deep Face Recognition: A Survey, 2018.

Given the breakthrough of AlexNet in 2012 for the simpler problem of image classification, there was a flurry of research and publications in 2014 and 2015 on deep learning methods for face recognition. Capabilities quickly achieved near-human-level performance, then exceeded human-level performance on a standard face recognition dataset within a three year period, which is an astounding rate of improvement given the prior decades of effort.

There are perhaps four milestone systems on deep learning for face recognition that drove these innovations; they are: the DeepID series of systems, VGGFace, and FaceNet. Let’s briefly touch on each.

The DeepID, or “Deep hidden IDentity features,” is a series of systems (e.g. DeepID, DeepID2, etc.), first described by Yi Sun, et al. in their 2014 paper titled “Deep Learning Face Representation from Predicting 10,000 Classes.” Their system was first described much like DeepFace, although was expanded in subsequent publications to support both identification and verification tasks by training via contrastive loss.

"**The key challenge of face recognition is to develop effective feature representations for reducing intra-personal variations while enlarging inter-personal differences. […] The face identification task increases the inter-personal variations by drawing DeepID2 features extracted from different identities apart, while the face verification task reduces the intra-personal variations by pulling DeepID2 features extracted from the same identity together, both of which are essential to face recognition.**"  
— Deep Learning Face Representation by Joint Identification-Verification, 2014.

The DeepID systems were among the first deep learning models to achieve better-than-human performance on the task, e.g. DeepID2 achieved 99.15% on the Labeled Faces in the Wild (LFW) dataset, which is better-than-human performance of 97.53%. Subsequent systems such as FaceNet and VGGFace improved upon these results.

FaceNet was described by Florian Schroff, et al. at Google in their 2015 paper titled “FaceNet: A Unified Embedding for Face Recognition and Clustering.” Their system achieved then state-of-the-art results and presented an innovation called ‘triplet loss‘ that allowed images to be encoded efficiently as feature vectors that allowed rapid similarity calculation and matching via distance calculations.

"**FaceNet, that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. […] Our method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches. To train, we use triplets of roughly aligned matching / non-matching face patches generated using a novel online triplet mining method**"  
— FaceNet: A Unified Embedding for Face Recognition and Clustering, 2015.

The VGGFace (for lack of a better name) was developed by Omkar Parkhi, et al. from the Visual Geometry Group (VGG) at Oxford and was described in their 2015 paper titled “Deep Face Recognition.” In addition to a better-tuned model, the focus of their work was on how to collect a very large training dataset and use this to train a very deep CNN model for face recognition that allowed them to achieve then state-of-the-art results on standard datasets.

"**… we show how a very large scale dataset (2.6M images, over 2.6K people) can be assembled by a combination of automation and human in the loop**"  
— Deep Face Recognition, 2015.

Although these may be the key early milestones in the field of deep learning for computer vision, progress has continued, with much innovation focused on loss functions to effectively train the models.

# Eigenfaces: A PCA Approach:

Face recognition is ubiquitous in science fiction: the protagonist looks at a camera, and the camera scans his or her face to recognize the person. More formally, we can formulate face recognition as a classification task, where the inputs are images and the outputs are people’s names. We’re going to discuss a popular technique for face recognition called eigenfaces. And at the heart of eigenfaces is an unsupervised dimensionality reduction technique called principal component analysis (PCA), and we will see how we can apply this general technique to our specific task of face recognition.

Before discussing principal component analysis, we should first define our problem. Face recognition is the challenge of classifying whose face is in an input image. This is different than face detection where the challenge is determining if there is a face in the input image. With face recognition, we need an existing database of faces. Given a new image of a face, we need to report the person’s name.

A naïve way of accomplishing this is to take the new image, flatten it into a vector, and compute the Euclidean distance between it and all of the other flattened images in our database.

There are several downsides to this approach. First of all, if we have a large database of faces, then doing this comparison for each face will take a while! Imagine that we’re building a face recognition system for real-time use! The larger our dataset, the slower our algorithm. But more faces will also produce better results! We want a system that is both fast and accurate. For this, we’ll use a neural network! We can train our network on our dataset and use it for our face recognition task.

There’s an issue with directly using a neural network: images can be large! If we had a single m\times n image, we would have to flatten it out into a single m\dot n\times 1 vector to feed into our neural network as input. For large image sizes, this might hurt speed! This is related to the second problem with using images as-is in our naïve approach: they are high-dimensional! (An m\times n image is really a m\dot n\times 1 vector) A new input might have a ton of noise and comparing each and every pixel using matrix subtraction and Euclidean distance might give us a high error and misclassifications!

These issues are why we don’t use the naïve method. Instead, we’d like to take our high-dimensional images and boil them down to a smaller dimensionality while retaining the essence or important parts of the image.

### Dimensionality Reduction:

The previous section motivates our reason for using a dimensionality reduction technique. Dimensionality reduction is a type of unsupervised learning where we want to take higher-dimensional data, like images, and represent them in a lower-dimensional space. Let’s use the following image as an example.

**Chart, scatter chart

Description automatically generated**

These plots show the same data, except the bottom chart zero-centers it. Notice that our data do not have any labels associated with them because this is unsupervised learning! In our simple case, dimensionality reduction will reduce these data from a 2D plane to a 1D line. If we had 3D data, we could reduce it down to a 2D plane or even a 1D line.

All dimensionality reduction techniques aim to find some hyperplane, a higher-dimensional line, to project the points onto. We can imagine a projection as taking a flashlight perpendicular to the hyperplane we’re project onto and plotting where the shadows fall on that hyperplane. For example, in our above data, if we wanted to project our points onto the x-axis, then we pretend each point is a ball and our flashlight would point directly down or up (perpendicular to the x-axis) and the shadows of the points would fall on the x-axis. This is a projection. We won’t worry about the exact math behind this since scikit-learn can apply this projection for us.

In our simple 2D case, we want to find a line to project our points onto. After we project the points, then we have data in 1D instead of 2D! Similarly, if we had 3D data, we want to find a plane to project the points down onto to reduce the dimensionality of our data from 3D to 2D. The different types of dimensionality reduction are all about figuring out which of these hyperplanes to select: there are an infinite number of them!

### **Principal Component Analysis:**

One technique of dimensionality reduction is called principal component analysis (PCA). The idea behind PCA is that we want to select the hyperplane such that when all the points are projected onto it, they are maximally spread out. In other words, we want the axis of maximal variance! Let’s consider our example plot above. A potential axis is the x-axis or y-axis, but, in both cases, that’s not the best axis. However, if we pick a line that cuts through our data diagonally, that is the axis where the data would be most spread!

**Chart, scatter chart

Description automatically generated**

The longer blue axis is the correct axis! If we were to project our points onto this axis, they would be maximally spread! But how do we figure out this axis? We can borrow a term from linear algebra called eigenvectors! This is where eigenfaces gets its name! Essentially, we compute the covariance matrix of our data and consider that covariance matrix’s largest eigenvectors. Those are our principal axes and the axes that we project our data onto to reduce dimensions. Using this approach, we can take high-dimensional data and reduce it down to a lower dimension by selecting the largest eigenvectors of the covariance matrix and projecting onto those eigenvectors.

Since we’re computing the axes of maximum spread, we’re retaining the most important aspects of our data. It’s easier for our classifier to separate faces when our data are spread out as opposed to bunched together.

(There are other dimensionality techniques, such as Linear Discriminant Analysis, that use supervised learning and are also used in face recognition, but PCA works really well!)

How does this relate to our challenge of face recognition? We can conceptualize our m\times n images as points in m\dot n-dimensional space. Then, we can use PCA to reduce our space from m\dot n into something much smaller. This will help speed up our computations and be robust to noise and variation.

Now that we have discussed PCA and eigenfaces, let us code a face recognition algorithm using scikit-learn! First, we will need a dataset. For our purposes, we will use an out-of-the-box dataset by the University of Massachusetts called Labeled Faces in the Wild (LFW).

In the first block of code, we will be using the Multi-layer perceptron classifier to classify the images in our dataset, and in the next, we will use the SVM algorithm with the RBF kernel.

**Results for MLP:**

precision recall f1-score support

Colin Powell 0.85 0.93 0.89 73

Donald Rumsfeld 0.79 0.82 0.81 38

George W Bush 0.88 0.91 0.90 151

Gerhard Schroeder 0.90 0.68 0.78 41

Tony Blair 0.83 0.74 0.78 39

accuracy 0.86 342

macro avg 0.85 0.82 0.83 342

weighted avg 0.86 0.86 0.86 342

**Results for RBF:**

precision recall f1-score support

Colin Powell 0.91 0.86 0.88 71

Donald Rumsfeld 1.00 0.57 0.73 35

George W Bush 0.81 0.99 0.89 164

Gerhard Schroeder 0.92 0.77 0.84 31

Tony Blair 0.93 0.63 0.75 41

accuracy 0.86 342

macro avg 0.91 0.77 0.82 342

weighted avg 0.87 0.86 0.85 342