COMPUTER VISION ASSIGNMENT 5

JUSTIN DE WITT* (21663904)

6 October 2022

1 PCA FEATURE VECTORS AND K-NN CLASSIFICATION

In this problem we would like to build a simple facial recognition system, using PCA feature vectors. To implement the system, we are using a *Georgia Tech* face database, composed of 750 photos, taken of 50 individuals.

1.1 Image Standardization

To determine the PCA feature vectors, we must first regularize the data. That is, we must guarantee that all images in the dataset are equal in size. There are two obvious approaches:

1. We can consider cropping each image about the image center. To implement this method, we must first determine the smallest image in the dataset. It was discovered that the smallest image in the dataset contains 156 rows, and 111 columns. It therefore makes sense crop a 154x110 rectangle out of the center of each image. This method may however cause information loss. It may be the case that the entirety of the face may not fit into the cropped result.
2. The second method to consider is strictly resizing each image. This method will not preserve the aspect ratio of each image. If some images contain a different row-to-column ratio, the faces may appear distorted if strictly resized to fit a mxn frame.

There are drawbacks associated with each of the mentioned methods. For example, consider the images shown in the following figures.

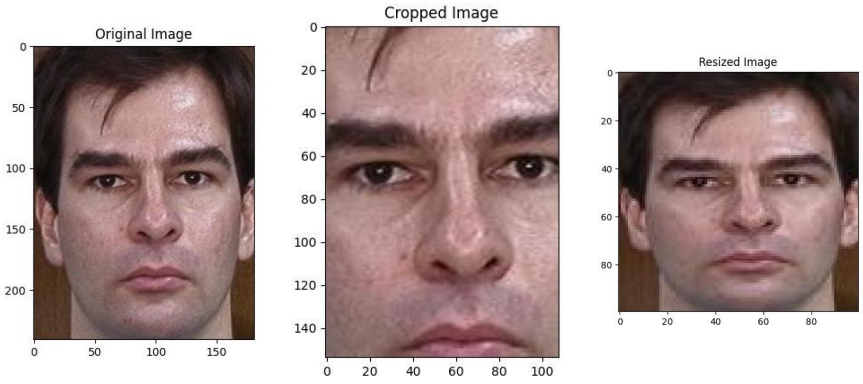

Figure 1: A side-by-side view of the unadjusted image, as well as the regularized version, using both methods.

As you can see in the figure above, the cropped version of the image contains information loss. We do not have the option to crop the image to a larger size, because this would imply cropping to a size which exceeds that of the smallest image. We therefore need to work with this information loss. Furthermore, we observe that the man's face appears wider in the resized image. This is because the image was re-sampled to fit a square. I have experimented with different aspect ratios, and did not observe any meaningful improvements. I have therefore simply opted to use a 100x100 square for this method. All of the images were subsequently regularized using each of the respective methods.

1.2 Sampling of the Hidden Test Set

To eventually assess the model's classification success, a set of regularized images must be set aside. For this, we randomly sample 5 images from each of the 50 photographed individuals. It is important that we remove these 250 images from the set. We do not allow these images to remain in the set used to train the model, otherwise our classification accuracy will be skewed.

1.3 Obtaining an Image Vector, the Average Vector, and a Matrix of Basis Vectors

In obtaining the values discussed in this subsection, we sample five photos (from the remaining 10) at random, for each of the 50 individuals in our dataset. We use this set of 250 photos for the derivation of the following values.

1.3.1 An Image Vector

We need to devise a method to transform an image, into a single Mx1 dimensional vector. To obtain this vector, we simply stack the columns of the image, into a single vector. Note that the image is in colour, therefore each pixel does not represent a single value, but rather three values. We subsequently also unstring each pixel into a vertical stack of its three coloured components. The result of a "unstrung" MxN dimensional image, is a $(3 * M * N)$x1 dimensional vector. We begin by "unstringing" each regularized RGB image into its corresponding image vector. When we refer to *each*, we are referring to the remaining 500 images, as we have removed 250 images to form the evaluation set.

1.3.2 The Average Vector

Now that a method has been devised to decompose an image into its corresponding image vector, we define the average vector as

$$\underline{\mathbf{a}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{f}_i$$

where $\mathbf{f}_i$ is the image vector corresponding to image i.

1.4 A Discussion as to the Relevance of the Average Vector

If you consider a hypothetical high dimensional coordinate system, where each image vector is drawn from the origin, it does not seem far-fetched to argue that the set of image vectors are contained in a high dimensional cloud, somewhere on this coordinate system. Recall that applying transformations (associated with matrix multiplication) is often much more simple, if the points of interest are centered about the origin. Consider applying a rotation for example. It is trivial to define a matrix which rotates about the origin, however defining a transformation which rotates around a point in M dimensional space is complex. Rotating a shape about

a point in M dimensional space need not be complex however, we simply translate to the origin, apply the trivial transformation, and translate the shape back to its original location. It is therefore useful to center the data of interest about the origin, so that a single defined transformation can be applied. Otherwise, a complex (and unique) transformation need be devised depending on the coordinate location of the field of interest. We can easily center our hypothetical vector cloud about the origin, by simply subtracting the *average* vector, from each vector in the cloud. In this context, the *average* vector is interpreted as a center of gravity, where all elements of the cloud contribute an equal weight. We show the image representation of this *average* vector in a proceeding section.

1.5 Determining the Basis Vector Set

As discussed in the previous subsection, we should center our high-dimensional datapoints about the origin. That is to say, we need to subtract the mean vector from each of the image vectors. This subtraction yields a new vector, $\mathbf{x}_i$, which matches the dimension of the image vector, however it has been translated towards the origin. To determine our basis vector set, we begin by packing all of our 250 $\mathbf{x}_i$ vectors into the columns of a matrix. We thereafter scale this matrix by constant factor, $\sqrt{250}$ in this case. We call this matrix X, and is size $(M * N * 3)$x250, where each regularized image is MxN. We now seek to obtain a basis which spans the column space of X. The singular values associated with the singular value decomposition (SVD) of X is helpful here. We plot the first 250 singular values in the following figure.

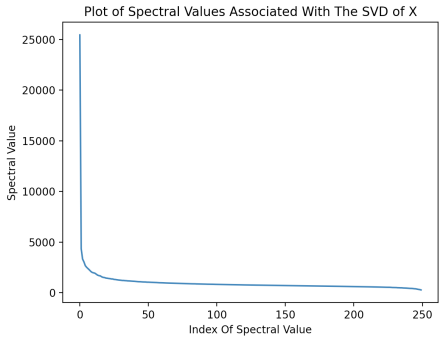

Figure 2: A plot of the 250 most significant singular values associated with the SVD of X.

We are interested in the retaining the principle components in order of significance. Singular values are directly related to eigenvalues, which give the *amount of variance* associated with each principle component. We wish to retain the components which explain most of the total variance, since variance is associated with information. If we consider the plot, it seems that most of the variance is explained by using the first 20 principle components, as thereafter each remaining principle component is associated with very little information. We therefore estimate that using approximately 20 columns of the U matrix (obtained from the SVD of X) span a sufficient low-dimensional subspace.

1.6 Dimensionality Reduction

We can now solve for the linear combination of basis vectors, coordinates, which best approximate each mean-centered image vector. That is, we can easily obtain $\underline{\mathbf{y}}$:

$$\begin{aligned} U_\alpha \underline{\mathbf{y}_i} &= \underline{\mathbf{x}_i} \\ \underline{\mathbf{y}_i} &= U_\alpha^T(\underline{\mathbf{x}_i}) \end{aligned} \tag{1}$$

because the transpose of U_α is its inverse (orthogonal columns). In this equation, $\underline{\mathbf{x}_i}$ represents any mean-centered image vector. The resulting coordinate vector $\underline{\mathbf{y}_i}$ represent the combinations of the basis vectors needed to (approximately) estimate our mean-centered vector $\underline{\mathbf{x}_i}$. Trivially, the dimension of our coordinate vector matches the quantity of vectors forming our basis, and our estimation accuracy is increased if we *increase* the size of the basis, after-all, this allows for the explanation of more variance. **However**, this is not to say that increasing the basis size will lead to better classification, more on that to come.

1.7 Reconstructed Image Estimations

We can easily reconstruct an image estimation by obtaining an $\underline{\mathbf{x}_i}$ estimate. We construct the $\underline{\mathbf{x}_i}$ estimation by summing the coefficient of each basis vector's contribution (coordinate), with that basis vector. The resulting vector sum is the $\underline{\mathbf{x}_i}$ estimate. Simply undo the mean-centering by adding the average vector $\underline{\mathbf{a}}$, and "re-stringing" the result into an image. A few figures of different basis sizes are shown below.

Basis Width: 40

Basis Width: 106

Basis Width: 172

Basis Width: 240

Figure 3: A comparison of image reconstruction using various basis sizes. The reconstructed estimation is shown to the right of the original. Note the direct correlation between basis size, and reconstruction accuracy. Once again, this later shown to not be directly correlated to eventual classification accuracy.

1.8 Eigenface Interpretation

The reconstruction of our approximated mean-centered vectors $\underline{\mathbf{x}_i}$, and the image estimations thereafter, poses an interesting thought. What if the coordinate used in the reconstruction were $[0_1, 0_2, ..., 1_k ..., 0_\alpha]$, where α is the number of vectors composing our basis. This is equivalent to asking the question: *What happens if we choose one of our basis vectors* ***as*** *a mean-centered estimate?* We can map each basis vector into an image (hopefully resembling a face). The composition of these faces should give us our estimation of the mean-centered version of the original face. For the purpose of visualization, I have interpolated the values in the eigenface estimation to fit the interval $[0, 255]$.

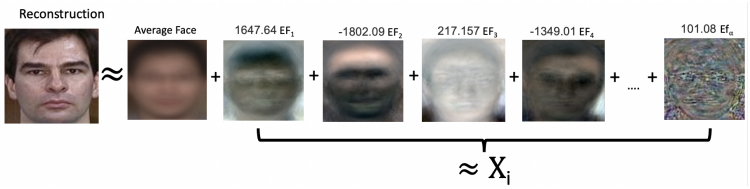

Figure 4: The four most significant eigenfaces, along with their respective coordinate contributions. Here a basis of 240 was used, for the purpose of pretty illustrations. We will determine later that a larger basis width does not yield optimal classification.

1.9 Classifying Feature Vectors Using K-NN

We now convert all the images in the dataset (including test data) to feature vectors. That is, we obtain the respective linear combination of basis vectors that best estimates the mean-centered image vector $\underline{x_i}$. Now that both the training set (10 images per person) and test set (5 images per person) have been converted to feature vectors, we need to label the sets respectively. We create two arrays, corresponding to the length of the test set, and training set respectively. Each index of the array, contains a numeric value associated with the "number" of the individual, the classifier will attempt to estimate these numbers correctly.

1.9.1 K-NN Classification

The K-nearest neighbor algorithm works by finding the k points in the sample set, that are nearest to a query point. Different definitions to define distance are used, but for the purpose of this paper, we use Euclidean distance. Once the set of k datapoints is obtained, the query point is classified as the majority class in the set. In the case of ties (even neighborhood size), *Scikit-Learn* uses a "first class to appear" approach, however the classes are returned in sorted order. Therefore, the final classification corresponds to the class of nearest distance.

1.10 Accuracy of K-NN for Various K

Here we plot the accuracy of the K-NN classification for various neighborhood sizes, and basis sizes. We obtain a plot for both the standardization methods discussed in section 1.1. Consider the following figures.

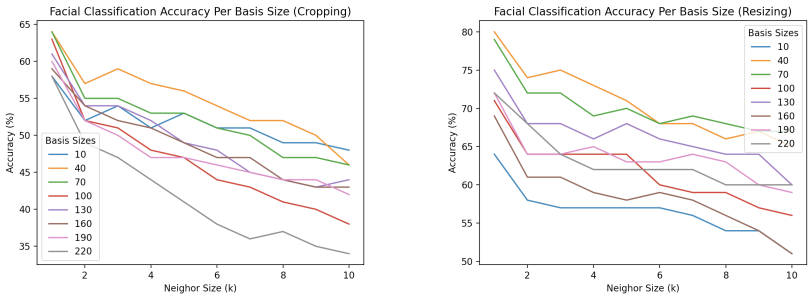

Figure 5: Classification accuracy using cropping and resizing as the standardization technique.

Based on the results from figure 5, we determine that a low neighborhood size (usually 1) and a basis width of approximately 25-40 yields the best classification accuracy. It is likely that the feature space is highly overlapping between classes. When we increase k, it seems like we are adding incorrect classes to our neighborhood, rather than multiple instances of the same class. This could be due to high variance in the feature space distribution of elements in the same class.

2 BAG-OF-WORDS FEATURE VECTORS AND SVM CLASSIFICATION

Similarly to the previous section, we wish to reduce the dimension associated with a piece of data, without losing valuable information which can be useful in distinguishing this datapoint from the rest. In this section we use a different approach. We attempt to decompose an image into a collection of *visual words*, where the vocabulary of visual words is obtained by considering the dataset as a whole. We perform classification from a datapoint's visual word composition.

2.1 Obtaining the Bag-Of-Words and Decomposing an Image Into a Normalized Histogram

For each of the images in both the training and testing dataset, we are provided a collection of 128-dimensional feature descriptor vectors. We would like to devise a method of capturing the information associated with each image, based upon this set of descriptor vectors. Using the descriptor vectors directly is not feasible, as the collection contains around 100 entries, each of which is 128-dimensional. However, it is not far-fetched to imagine that images of the same subject may contain similar features. That is, similar images should contain overlapping feature descriptors. Similarly to the "high dimensional plot" described in section 1.4, it is logical to assume that similar images (or components thereof) are situated in relatively near proximity to one another. We can run a clustering algorithm which attempts to find β high-dimensional clouds, which minimize variance between elements of the cloud, but maximize the distance between clouds. These "clouds" are referred to as clusters. We can attempt to identify clusters based upon the **entire** set of descriptor vectors, across all images. Once a set of clusters (bag-of-words) is located, we can once-more propagate the descriptor vectors associated with an image through the model, however this time we note the cluster to which each vector mapped. We assign each of the β identified clusters, a number (indexed from $[0, \beta)$), and note to which clusters the descriptor vectors of each image map. In this application, each cluster corresponds to a visual word, and we assume that similar images will contain similar frequencies of visual word mappings. A set of these mappings is displayed in the following figures.

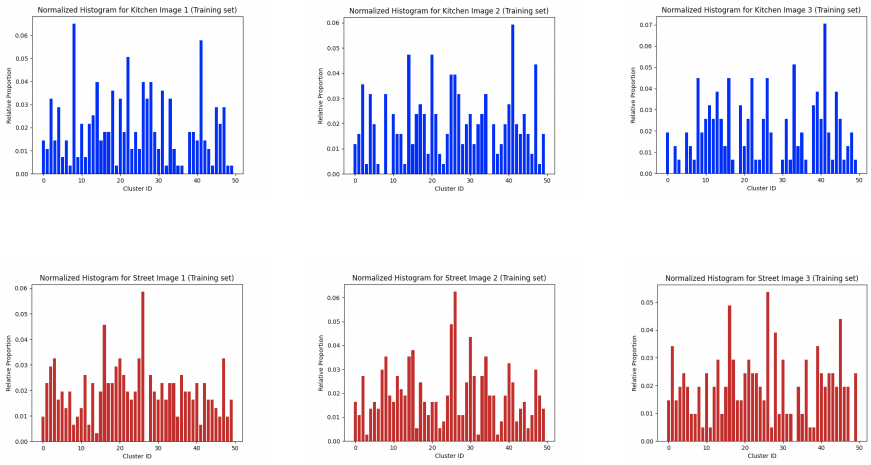

Figure 6: A comparison of the normalized histograms associated with the first three kitchen images, and first three street images respectively. Notice that the normalized histograms associated with kitchens and streets appear different, classification success relies on the consistency of this difference.

We say *normalized* because rather than keeping the y-axis a frequency, we have divided each frequency by the length of the descriptor vector set. The histogram now more closely resembles a distribution, similar to a 1-observation discrete *probability density function*. Using these normalized histograms we have essentially captured the information of an image, but in a much smaller package- only 50-dimensions in this case. We can now perform classification by training a model to recognize the differences and similarities between normalized image histograms.

2.1.1 Clustering Dimensionality Complications

As mentioned, we would like to partition the set of all descriptor vectors (across all images) into a set of 50 clusters. Unfortunately, there are **many** descriptor vectors (each 128-dimensional) that our K-Means algorithm must cluster. K-means is an iterative algorithm that plots the datatset (or subset thereof) and attempts to find centroid locations which best identify center-of-gravity locations relative to the points surrounding the centroid. One can image that repeating this process for $5,000$ iterations, as used in this implementation, is slow. We can consider sampling a random subset of the training dataset on each iteration, and clustering based on this set. In theory, if we repeat this over many iterations, we should approximate the true centroids, with much less computational effort per iteration. Mini-batching does introduce stochasticity into the model though, as the elements of the batch are chosen randomly during each iteration. In this assignment, I explore this hypothesis, using *batch sizes* of [50, 350, 1024, 5000, ALL]. Don't worry, I've serialized my models so that they need not be retrained, and can be easily used later.

2.2 Linear One-vs-One Support Vector Machine Classification

To classify unseen test images, we train an array of *Support Vector Machines* (SVMs). Each support vector machine acts as a binary, linear classifier. That is, each SVM in the array can only distinguish between two classes. There are 9 classes in our dataset, so if we wish to create a SVM for every combination of class-pairs, we need 36 SVMs in the array. Each SVM in the array is trained only on data corresponding to the two classes which is attempts to classify. When we wish to classify a query-point, we allow each SVM in the array to provide an opinion. Our classifier acts as a democracy, the majority wins. This is a successful strategy, because we are guaranteed that 8 of the machines have been explicitly trained to recognize the true class associated with any query point. If we assume these 8 classifiers perform acceptably, they will "out vote" those who have uneducated opinions (The remaining 26 SVMs).

2.2.1 Hyper-Parameter to the Support Vector Machine

Over-fitting to the training data is not far from the Achilles heal of machine learning algorithms. We need to strike a favorable trade-off between getting a high classification accuracy on the training data, but not pulling the decision boundary towards potential outliers. There exists a desirable balance between maximally fitting the training data, and ignoring peculiar points in the training set. In the context of support vector machines, we can control the hyper-parameter C. The choice of regularization parameter C determines the degree to which the SVM attempts to fit the training data, the relationship is direct. In our implementation, we will experiment with different values for C, as well as different batch-sizes (as mentioned previously).

2.2.2 Running the Simulation

To obtain our results, we need to first convert our set of test data into feature vectors using the bag-of-words approach mentioned in section 2.1. Thereafter we

can propagate the feature vectors of each image through the SVM array, and obtain our estimated classification. The simulation procedure continuous as follows:

1. Obtain a set of all batch-size, C, pairs. Use these parameters to initialize a K-Means model, and SVM array.
2. Classify all feature vectors associated with the test set using the obtained models.
3. Count the number of correct classifications compared to the length of the test set, this metric represents classification accuracy.
4. Repeat until we have identified the optimal batch-size, C, pair.

2.2.3 *Identifying the Optimal Classifier Parameters*

We have implemented the procedure, and obtained a final classification accuracy score associated with each batch-size, C, pair. I find it best to display these results to the reader in the form of a heat-map.

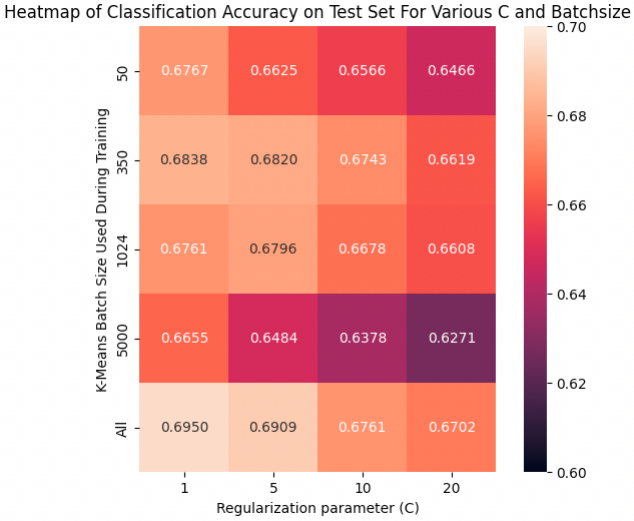

Figure 7: A heat-map illustrating the performance of each batch-size, C, pair. The value in the center of each cell is overall classification accuracy, displayed as a decimal.

As suggested by the figure, the optimal configuration is a rather flexible $C = 1$, and a batch-less K-Means model. It may not always be computationally feasible to perform K-Means on the entire dataset, and in such a scenario, it seems we achieve better results for larger batch-sizes. I believe the results for batch-size 5000 are unlucky, a result of random initialization. Furthermore, it appears in this application, a flexible choice for C always performs better.

2.2.4 *Confusion Matrix Using The Identified Parameters*

Now that we have identified a set of parameters which yields favorable classification performance, we will repeat the classification simulation once more using these parameters. However, during this simulation, we will track the predictions of our model, and the correct classifications. We would like to determine if our model has a tendency to confuse certain classes, or is unusually poor at classifying certain classes. We use a common machine learning figure, the *confusion matrix*, for this visualization. The confusion matrix is simply a grid-based plot where the x-axis

corresponds to the predictions of the model, and the y-axis being the correct classification. Trivially, we would prefer for the model to classify everything correctly, so a perfect confusion matrix contains entries only along its main diagonal.

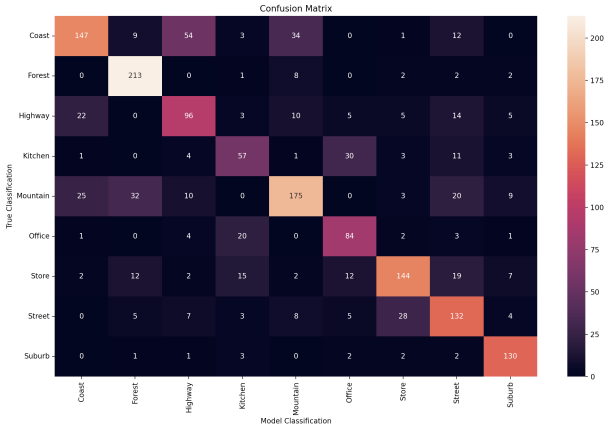

Figure 8: A graphical representation of the obtained confusion matrix, using the optimal parameter configuration. It appears that our model often confuses a scenic photo of the coast, with a highway...

2.2.5 Photos and Videos of Correct and Incorrect Classifications

For those who prefer videos, the following links contain short videos where the successes and shortcomings of the classifier can be celebrated.

Press Here for a video montage of correct classifications.

Press Here for a video montage of incorrect classifications.

Otherwise, a few images are included below. You will notice that sometimes the model makes mistakes that are understandable, as seen in the figure. The videos reveal that other mistakes are a rather blatant, but perhaps further tuning efforts can be committed in the future.

Figure 9: A few of the mistakes made by the model. Notice that these mistakes seem somewhat understandable. There is indeed a mountain in the coastal photo, and that office is not the most "office like".

Figure 10: A few of the correct classifications predicted by the model. For more, view the video linked above!

2.3 Plotting of Incorrect Classifications

It may be interesting to investigate the circumstances under which the model incorrectly classified the query image. Consider the following figures:

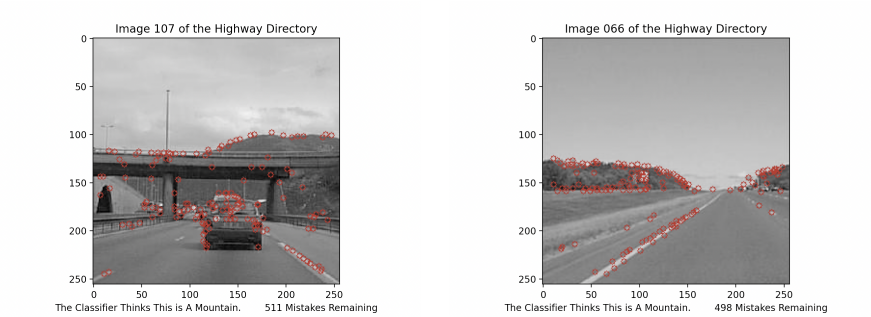

Figure 11: Image locations of the provided descriptor vectors, on which the model bases its classification.

In the case of failing to classify some "highway" scenes correctly, you will notice that a decent subset of the provided descriptors are located in a semi-mountainous region in the background of the highway. It could be that these descriptors are what is causing the incorrect classifications to some extent. A similar trend is observed for the other images, though these photos best illustrate my observations. I am unsure as to how we can combat this *"issue"*. Maybe it is possible to devise a SIFT based algorithm that attempts to identify feature matches that are more evenly distributed across the image. This reduces the likelihood of matches corresponding to certain features dominating, and potentially skewing, the final classification.