Introductory Applied Machine Learning: Assignment

School of Informatics, University of Edinburgh

Instructors: Victor Lavrenko and Charles Sutton

Handed out 2 Nov 2011, due by **1600** on **Tue 22 Nov 2011**.
Hard copy **and** electronic submission required. The time of the deadline will be strictly enforced.

Remember that plagiarism is a university offence. Please read the policy at http://www.inf.ed.ac.uk/teaching/plagiarism.html .

Marking Breakdown

A results/answer correct plus extra achievement at understanding or analysis of results. Clear explanations, evidence of creative or deeper thought will contribute to a higher grade.

B results/answer correct or nearly correct and well explained.

C results/answer in right direction but significant errors.

D some evidence that the student has gained some understanding, but not answered the questions properly.

E/F/G serious error or slack work.

Mechanics

You should produce a word processed report in answer to this assignment (e.g. with LaTeX).

- **postscript or pdf** formats are acceptable for the report, other formats are not.
- you need to submit this report as a hard copy to the ITO **and** electronically as described below.

In part E you are also required to produce a file `iaml_assignment.res`. For the electronic submission place this file and your report in a directory called `iamlans` and submit this using the `submit` command on a DICE machine. The format is

```
submit iaml 1 iamlans
```

You can check the status of your submissions with the `show_submissions` command.

NOTE: Your electronic submission will **not** count if you do not submit a hard copy of your report to the ITO.

Late submissions: The policy stated in the School of Informatics MSc Degree Guide is that normally you will not be allowed to submit coursework late. See `http://www.inf.ed.ac.uk/teaching/years/msc/courseguide10.html#exam` for exceptions to this, e.g. in case of serious medical illness or serious personal problems.

Collaboration: You may discuss the assignment with your colleagues, provided that the writing that you submit is entirely your own. That is, you should NOT borrow actual text from other students. We ask that you list on the assignment sheet a list of the people who you've had discussions with (if any).

Description of the dataset

This assignment is based on the the Extended Yale Face Database B. The extended Yale Face Database B contains 16,128 images of 38 human subjects and around 64 near frontal images under different illuminations per individual. Our task in this assignment will be to perform face recognition on the database. Face recognition is the process of matching a given face to one of the many the faces that are known to the database. Face recognition is made difficult by factors such as the great variability in head rotation and tilt, lighting intensity and angle, facial expression and aging.

To save you time and to make the problem manageable with limited computational resources, we have created a smaller dataset by selecting a random subset of images from the original dataset. The dataset now contains 10 distinct human subjects with 20 images per subject in the training set. Our task will therefore be multi-class classification with 10 classes (one per person). As well as being resized to 32×32 pixels the images have also been cropped to remove background, aligned and converted to grayscale (256 grey levels per pixel). This smaller dataset has been converted to the sparse arff format[1], and can be thus used directly to perform our experiments in WEKA.

There is, however, a potential caveat: The processed dataset has been prepared by a busy and heavily underpaid teaching assistant who might have been a bit careless when preparing the dataset. You should keep this in mind and be aware of anomalies in the data when answering the questions below.

For this assignment we provide three data sets: a training set, a validation set and a test set. The training set and validation set contain valid labels. In the test set the labels are missing. In the first four parts of the assignment you will only be using the training and validation data. The files are available from the IAML website.

Important Instructions

(a) In the following questions you are asked to run experiments using WEKA. The WEKA version installed on DICE is **Version 3.6.2**. If you are working on a machine other than DICE (e.g. your laptop), please make sure that you download and install the **same** version. This is important as your results need to be reproducible on DICE.

(b) In some of the questions you are asked to perform experiments in the WEKA *Experimenter*. Instructions on how to use the *Experimenter* are given at the end of the assignment.

(c) In many cases, the WEKA *Explorer* allows you to modify the random seed that will be used. Just using the default seed is fine. If you do change the seed you need to report the seed you have chosen.

(d) In some of the questions you are asked to run some commands in *Matlab* for the purposes of visualization of the data. If you are unfamiliar with Matlab please see the end of the assignment where you will find explicit instructions on running the required commands within Matlab. *Important:* There are a limited number of Matlab licenses in the University, so please try to keep the duration of your Matlab session to a minimum. Ensure that you close Matlab once you are finished so that another student can use the license.

(e) You may find that WEKA crashes with an out-of-memory exception. If this should occur, refer to the instructions in IAML lab 1 in order to run WEKA with a larger memory allocation.

(f) The .arff files that you will be using are available from the IAML website. We also provide you with the Matlab .mat files for displaying the training dataset faces.

(g) You should aim for answers with **less** than 150 words per individual question. NOTE: you may **lose points** for answers longer than 250 words.

[1] See `http://weka.wikispaces.com/ARFF+(stable+version)#Sparse ARFF files` for a description of the sparse arff format.

Part A: Exploration of the dataset [20%]

Load the dataset `train_faces.arff`. Your first task is to get a feel for the data that you will be dealing with in the rest of the assignment. Unless stated otherwise, we will use only the training set and we will use cross validation (CV) on this dataset to evaluate the classifiers. In order to reduce computation time you should use 5-fold cross validation unless explicitly specified otherwise (the default setting is 10).

(i) Train and evaluate different classifiers on the dataset using 5-fold CV. Try a Naive Bayes (`NaiveBayes`) classifier, a decision tree (`J48`) and a linear Support Vector Machine (`SMO`). For now, use the default settings for the classifiers. We are not interested in optimizing their parameters, we just want to get a first idea of the dataset. Write down the accuracy (percent correct, PC) of the different classifiers.

(ii) We will explore the dataset using *clustering*. Ideally we expect that the clustering will assign similar faces to the same cluster. To perform the clustering we will use the unsupervised attribute filter *AddCluster* which adds the cluster label to our training set as an additional feature. Before selecting this filter you will need to specify *No class* in the drop down list on the *Preprocess* tab (located just to the left of the "Visualize All" button). Change the options of the AddCluster filter to use the *EM clustering* algorithm. What do you think might be a sensible choice for the number of clusters?

Once you have decided on the number of clusters *Apply* the filter to the dataset. This will cluster the training set, adding the cluster label of each instance as an additional attribute. Once clustering has completed you can view the histogram of the cluster labels in the bottom, right-hand corner of the *Preprocess* tab. Do you notice anything unusual? Relate your observations to the classification performance in part (i).

HINT: You can visualize instances belonging to a particular cluster by selecting *Edit* on the Preprocess tab to view the dataset itself. This will bring up a window that shows the full dataset (one row per data point, one column per attribute). The cluster labels are in the last column. The row number is the instance ID that can be used with the supplied `draw_faces.p` Matlab code and the `train_faces.mat` Matlab data file to how the face that corresponds to that instance (please see the end of the assignment for a description on how to run this command in Matlab).

(iii) Based on what you found out about the data in the previous questions, clean up the training set as you see fit. Describe what you have done and explain why.

HINT: The `RemoveWithValues` filter could be useful for this purpose (this is an unsupervised instance filter in the *Preprocessing* tab).

(iv) Retrain and evaluate the classifiers that you have been using in (a) using 5-fold CV. What is their performance on the modified dataset? What is a simple baseline against which to compare the performance of the three models? *Important:* You will need to remove the Cluster Id attribute from your training set before re-training your classifiers.

For the rest of the assignment you *must* use the cleaned-up dataset we have provided: `train_faces_clean.arff`. You can download this from the IAML website.

(v) It can be instructive to look at examples where the classifier works well or where it goes wrong. To do so choose again the linear Support Vector Machine (`SMO`) classifier. Load the clean training set (`train_faces_clean.arff`), set the test options to *Supplied test set* and choose the *validation set* `val_faces.arff`. To see the classifier output for each data point in the test set click on *More options* and check the *Output predictions* box. Now train and evaluate the classifier. When you scroll up in the *Classifier output* text field you will see a large table that contains the *Predictions on test set.* Each row in this table corresponds to a data point in the validation set (`val_faces.arff`). The probabilities output by the classifier that the image contains the face of a particular person are the values in the last ten columns of the table. The * preceding one of the probability values indicates the class the data point has been assigned to (always the class with largest probability). Across in the *error* column indicates a misclassification. The instance number corresponds to the face in the `val_faces.mat` file.

Identify faces that belong to each one of the following two groups: (a) misclassified (b) correctly classified. Select *two* images from each group and include them in your report. Comment on whether the kinds of images seen in the two different groups accord with your expectations. What appears to give the classifier the most trouble?

Part B: Feature Selection & Engineering [20%]

Feature selection and feature engineering are important aspects of machine learning in practice. In the following section we want to assess the usefulness of the features and the impact of feature engineering on the classification task. All experiments should be performed on the training data using *5-fold CV* and the default options. *Important:* For this Section ensure you are no longer performing evaluation using the validation dataset.

(i) We will now look at the Naive Bayes model in more detail. What kind of distribution is used to model the attributes? Do you think this is a sensible choice? Why?

We will explore the effect discretization has on the performance of the Naive Bayes model. Reload the clean training dataset (`train_faces_clean.arff`). Discretization is the process of converting numeric attributes into nominal attributes by assinging the numeric attributes to a number of distinct ranges. We will use the unsupervised discretization provided by WEKA. From the Preprocess tab select *Filters* `->` *Unsupervised* `->` *Attribute* `->` *Discretize.*

We will experiment with values of 5, 10, 20, 40, 60, 80 and 100 for the number of bins using *equal frequency binning* (set `useEqualFrequency` to true). This flavour of discretization allows the intervals (bins) to be of different sizes, choosing them so that the same number of training examples fall into each one. What do you think are the important considerations when choosing the number of bins?

Apply the filter to the dataset and re-run your Naive Bayes model for each change of bin number. Record the accuracy of the classifier in each case (don't forget to re-load your dataset `train_faces_clean.arff` for each bin number). Report the bin number that leads to the best performance. Is there a salient difference in performance of the Naive Bayes classifier compared to Part A? Can you explain the performance of your model on the discretized data?

(ii) We will now assess the *usefulness* of the features using the attribute evaluator `InfoGainAttributeEval` (in the *Select Attributes* tab). How does this 'attribute evaluator' work? Reload the clean dataset `train_faces_clean.arff`. Apply the attribute evaluator `InfoGainAttributeEval` and remove all the attributes (pixels) that have 0 information gain.

HINT: To remove the attributes go to the *Preprocess* tab and select the `AttributeSelection` filter. Click on it and change its options to `InfoGainAttributeEval` for the evaluator and `Ranker` for search. This filter will re-order the attributes using the information gain criterion. You can then use the `Remove` filter and specify the appropriate range of attribute indices to be removed, being careful not to remove the class.

For your convenience we also provide the reduced data set as `train_faces_clean_best.arff`.

(iii) It is instructive to visualize the position of the remaining pixels superimposed on a given face. For your convenience we provide you with Matlab code `draw_pixels.p` and the associated `train_faces_clean.mat` Matlab data file to visualize a face with superimposed pixels (please see the end of the assignment for a description on how to run this command in Matlab). Do the location of the pixels make sense for the face recognition task? Include an image plot with your report to illustrate your explanation. What is a potential problem with the information gain criterion? HINT: Think about the case where you use it to select a number of features at once, rather than a single feature.

(iv) Retrain and evaluate the Naive Bayes, Decision tree and linear Support Vector Machine using the reduced dataset (`train_faces_clean_best.arff`). Report their performance and compare it to the performance with the *clean* dataset from part A (`train_faces_clean.arff`). Explain the performance of each classifier on the reduced dataset.

Part C: More Advanced Feature Selection [25%]

We will use the `train_faces_clean.arff` file for this section of the assignment. By now, we have reduced the dimensions of our dataset to 349. However, we expect the faces to lie in a lower-dimensional manifold and want to examine the representation we get by applying Principal Components Analysis (PCA). PCA maps the data into a new space by effectively rotating the basis vectors of the input space to the directions with the highest variance. In the following section, we will assess the impact of this mapping to the classification task and the separability of the data in the PCA space. As in the previous section, all experiments should be performed on the training data using 5-fold CV and the default options.

(i) Load the dataset `train_faces_clean.arff`. Perform PCA using the attribute evaluator `PrincipalComponents` (in the *Select attributes* tab). Change the `varianceCovered` option to 1, in order to get the Eigenvalues for all the Eigenvectors of the input space. Plot the Eigenvalues in descending order. What do you notice?

(ii) Now go to the *Preprocess* tab and select the `AttributeSelection` filter. Click on it and change its options to `PrincipalComponents` for the evaluator and `Ranker` for search. This filter will map our dataset into the principal components and will use as many Eigenvectors as to retain 95% of the variance in the data. How many Eigenvectors does the resulting representation have? Retrain the Naive Bayes, Decision tree and linear Support Vector Machine classifiers and report their performance. How does it compare to your results from part A? Explain the performance of each classifier.

Save the resulting dataset as `train_faces_clean_pca.arff` you will need it in Part D.

(iii) In performing PCA on our face data and projecting the data onto the Eigenvectors, we have essentially applied the *Eigenfaces* approach to face recognition, where the Eigenfaces are the Eigenvectors we obtain after the PCA transformation. The projection operation characterizes an individual face by a weighted sum of the Eigenface features, and so to recognize a particular face it is necessary only to compare these weights to known individuals. Examine the Eigenfaces that together capture 95% of the variance in the data. What do the Eigenfaces represent? Produce an image plot of the top two Eigenvectors, give their corresponding Eigenvalues and provide a description of their characteristics.

HINT: It might help to visualize the top Eigenfaces using `draw_eigenfaces.p` and associated data file `train_faces_clean.mat`.

(iv) In principle we can now effectively reconstruct any face in our database using the Eigenfaces as our basis or "templates". In this question we will investigate how many principal components (Eigenfaces with the largest Eigenvalues) are required to obtain human-recognizable reconstructions. We will use the supplied Matlab code `reconstruct_face.p` to draw a face from our original training dataset using a weighted combination of Eigenfaces from our "face space". Vary the number of principal components used for the reconstruction and describe what is happening as the number of components are gradually increased. Include a suitable selection of images to illustrate your explanation. What is the *minimum* number of principal components required to produce a face that is recognizable as the original face?

(v) Under the WEKA *Preprocess* tab remove from your PCA projected dataset the top four most *significant* principal components i.e. those Eigenfaces which you found to have the four highest corresponding Eigenvalues in your answer to (i). Retrain the Naive Bayes classifier and report the result. Given that we are throwing away the top four most significant principal components that capture a large proportion of the variance in the dataset we might well expect the classifier accuracy to decrease. Do you agree or disagree? Explain your reasoning.

(vi) Finally we will study the influence of the Eigenfaces dimensionality on the performance of the linear SVM classifier. We will train a linear SVM, keeping its parameters constant and we will vary the number of Eigenfaces used to construct the "face space". Reload `train_faces_clean.arff`, apply PCA to the dataset and try 20, 40, 80, 120, 160 and 200 of the top Eigenfaces (Eigenfaces with the largest Eigenvalues) recording the percent correct (PC) in each case. Plot and interpret your results. What do you notice? Can you come up with an explanation for the performance of your classifier?

HINT: It might be easier to set the `varianceCovered` option to 1 to obtain all 1024 Eigenfaces. You can then remove all but the required number of Eigenfaces in the WEKA *Preprocess* tab. Don't forget to reload the dataset `train_faces_clean.arff` afresh after changing the number of Eigenfaces. The *Invert* button provided by WEKA can be useful when you wish to throw away a lot of data points.

Part D: Choosing representation and classifier [15%]

In the next section we are interested in more rigorously comparing the performance of classifiers in different datasets. So far, we have used the percent correct, averaged across CV folds, as a performance measure. This is not sufficient for a rigorous comparison. We also need an estimate of the variability of the classifier performance across data sets. One way to quantify this variability is to consider the standard deviation of the PC value across CV folds and different runs. Since this is not easy to obtain in the *Explorer* and since the *Explorer* is not very convenient for large-scale experiments we will use the *Experimenter* where indicated below. When reporting the classifier performance you should consider the average PC as well as the standard deviation. A short description of how to use the *Experimenter* can be found at the end of the assignment.

(i) First, we wish to explore the impact of different data representations on the classification task. To investigate this we will use three different representations from the previous sections to train a Naive Bayes model, a Decision tree and a linear SVM. In the *Experimenter*, select the datasets `train_faces_clean.arff`, `train_faces_clean_best.arff` and `train_faces_clean_pca.arff` and the classifiers `NaiveBayes`, `J48` and `SMO` with default parameters. Use 5-fold cross-validation and choose to repeat the experiment 5 times (change the *Number of repetitions*). Report the average percent correct (PC) and the standard deviation of each classifier for each of the datasets.

For which dataset(s) does the SVM perform significantly better than Naive Bayes with 95% confidence? Is any of the datasets significantly better than the other datasets for Naive Bayes with 99% confidence? Similarily are any of the datasets significantly better than the other datasets for the Decision tree with 99% confidence? What dataset would you therefore select for the rest of your experiments in light of these results?

(ii) Next, we wish to explore the impact of the amount of training data on the classification performance. To investigate this we will train a linear SVM using nested subsets of the training data with different sizes. We will train the classifier using the `train_faces_clean.arff` dataset and evaluate them on the validation dataset `val_faces.arff` (instead of cross validation on the training set). You should perform this experiment in the *Explorer*. Use 5%, 10%, 35%, 50%, 65%, 80% and 100% of the data points in the training set. Plot percentage correct as a function of the size of the training set. Interpret your results. Do you think we would get better classification performance if we had more examples in the training dataset?

HINTS: To remove data points first randomize the dataset (using the `Randomize` instance filter) and then remove different percentages of the data points (using the `RemovePercentage` instance filter). Make sure you use the same randomization (same seed for the `Randomize` filter) for different subset sizes so that the subsets are nested. Report the seed that you have chosen to randomize the dataset. In order to evaluate the classifiers on the validation set (instead of cross-validation on the training set) choose 'Supplied test set' in the 'Test options' box in the *Classify* tab and set the test set as appropriate.

(iii) Finally we will now combine two different feature representations to determine if there is any benefit to feature combination. We will train the classifier using the `train_faces_clean.arff` dataset with evalution being performed on the validation dataset `val_faces.arff` (instead of cross validation on the training set). You should perform this experiment in the *Explorer*.

Firstly we will determine the effect of PCA on the performance of the Naive Bayes classifier on the validation dataset. As we are now using two different datasets (training on the training dataset, testing on the validation dataset) we will need to use a `FilteredClassifier` to ensure both datasets are compatible after the PCA transformation has been applied. The `FilteredClassifier` can used to encapsulate an attribute selection process with a classifier. To do this choose the *meta classifier* FilteredClassifier. Within the FilteredClassifier's options choose `NaiveBayes` as the classifier and the unsupervised attribute filter `PrincipalComponents` (using the default options). Report the performance of the model.

We will now combine PCA with cluster labels. To do this we will *nest* two FilterClassifiers. Choose the meta classifier FilteredClassifier. Within the FilteredClassifier's options choose another FilteredClassifier as the classifier and the unsupervised attribute filter *AddCluster* as the filter (setting the clusterer option to EM with an appropriate number of clusters). Within the second (nested) FilterClassifier's options select `NaiveBayes` as the classifier and the unsupervised attribute filter `PrincipalComponents` (with default options). You have now created the following data pre-processing chain: the training and validation datasets will first have cluster labels added followed by the application of PCA.

Record the performance (percent correct PC) for the Naive Bayes classifier and compare it to the performance you obtained using just the PCA representation on the validation dataset. How has the performance been affected by the addition of cluster labels to the PCA representation? Another student tells you that they think there's no way adding cluster label as features can help, because they contain no information that was not in the original features. Do you agree? Why or why not?

NOTE: As you can imagine it is fully possible to have even more pre-processing steps before the evaluation of the classifier itself by simply continuing to nest the FilteredClassifiers in the manner we have just performed.

Part E: Mini-challenge [20%]

In this final part of the assignment we will have a mini classification challenge. Using the data provided you are asked to find the best classifier for the face recognition task we have been dealing with. You can apply any preprocessing steps to the data that you think fit and employ any classifier you like (even those we have not used in the assignment). The only restriction is that the classification steps must be performed in WEKA. Preprocessing of the data may be performed within or outwith WEKA, at your option. If you perform the data preprocessing outside of Weka you are required to describe in detail how your custom preprocessed dataset can be reproduced. Please note that the thoroughness of the exploration and the resulting discussion is just as important as the final result.

We provide you with three data sets, with two variants of each (one that has and one that has not had its attributes filtered by information gain):

1. A training set:
   - `train_faces_clean.arff`: contains all 1024 grayscale pixels representation taking values in the range $\{0, \ldots, 255\}$.
   - `train_faces_clean_best.arff`: contains only the top 349 grayscale pixels (taking values in the range $\{0, \ldots, 255\}$) selected by information gain.
2. A validation set:
   - `val_faces.arff`: contains all 1024 grayscale pixels representation taking values in the range $\{0, \ldots, 255\}$.
   - `val_faces_best.arff`: contains only the top 349 grayscale pixels (taking values in the range $\{0, \ldots, 255\}$) as selected by information gain.
3. A test set:
   - `test_faces.arff`: contains all 1024 grayscale pixels representation taking values in the range $\{0, \ldots, 255\}$.

- `test_faces_best.arff`: contains only the top 349 grayscale pixels (taking values in the range $\{0, \ldots, 255\}$) selected by information gain.

You have already used the training set and the validation set in the questions above. You should use the former two for training and evaluating your models (as you see fit).

If you train a model on one variant of the datasets you must also validate and test it on the same variant. For example a model trained on `train_faces_clean_best.arff` must be validated on `val_faces_best.arff` and tested on `test_faces_best.arff`. Otherwise WEKA will complain that the *train and test set are not compatible*. If you wish to apply nested data transformations and still have the validation and testing sets be compatible with your training dataset you can use the the meta classifier `FilteredClassifier` as you did in Part D. The `FilteredClassifier` can used to encapsulate an attribute selection process with a classifier. The attribute selection and the classifier are learned on the training data; any validation/test instances have the identical transformation applied before being passed to the classifier for prediction. This ensures both datasets are compatible. Multiple levels of transformation can be achieved by nesting multiple `FilteredClassifier`s.

Once you have chosen your favourite model (and preprocessing steps) you should apply it to the test set (for which no labels are provided). Classify the data points in the test set and submit the classification results as part of your answer. Your results will be evaluated in terms of the percentage correct. You also need to submit a brief description of your approach, and a short explanation of why you chose it.

HINT: Throughout the assignment we have used use the default settings for the classifiers. Note that feature engineering, feature combination and model parameter optimization can significantly improve model performance.

How to submit the results: Load the data set that you would like to train your final classifier on. Choose "Supplied test set" in the "Test options" box and select `test_faces.arff` (or an appropriately preprocessed version thereof). Click on "More options" and make sure that the "Output predictions" box is checked. Then start the training and evaluation. When the training/evaluation is finished right click on the entry in the result list and choose "Save result buffer". Save the result buffer as `iaml_assignment.res`. Again: *Make sure that the "Output predictions" box in the "More options" dialog is checked - otherwise the output buffer will not contain predictions for the individual data points and your results cannot be evaluated!!* The result summary will not be meaningful for the test set because no labels are provided!

HINT: The "Re-evaluate model on current test set" option, which you get when you right-click on an entry in the result list, allows you to check multiple test sets with the same classifier.

Using Matlab

You will be required to run two Matlab scripts to visualize the data in the assignment. You can follow these instructions to run a script within Matlab.

Important: There are a limited number of Matlab licenses in the University, so please try to keep the duration of your Matlab session to a minimum. Ensure that you close Matlab once you are finished so that another student can use the license.

Starting Matlab

1. Open up a Linux command prompt and type `matlab`. This will open the Matlab environment on your computer.

2. You will now need to point Matlab to the location of the assignment Matlab files. To do so, click *File* `->` *Set Path*`->` *Add with Subfolders*. Browse to the folder that contains the Matlab code for the

assignment. Once found, highlight the folder and click Ok. Click Save and click No to saving the `pathdef.m` to another directory. Click close. Matlab will now be able to find your code whenever you wish to run one of the supplied Matlab scripts.

Displaying Faces

1. The supplied Matlab code `draw_faces.p` can be used to visualize faces from the training dataset.
2. To run this code in Matlab you will need to type the command in the Matlab command window. The parameter specification is:

   ```
   draw_faces(DATA_FILE,[INSTANCE_IDS])
   ```

   Where `DATA_FILE` is the full path to the location of the `train_faces.mat` on your machine. `INSTANCE_IDS` is a vector of instance ids corresponding to the faces which you wish to display (up to a maximum of 5 faces can be displayed at any one time).
3. For example if you saved the `train_faces.mat` file in your home directory and wished to display the 5 faces with instance ids 1,6,44,51,3 an example invocation would be:

   ```
   draw_faces('~/train_faces.mat',[1,6,44,51,3])
   ```

Displaying Superimposed Pixels

1. The supplied Matlab code `draw_pixels.p` can be used to visualize the best pixels (as selected by information gain) superimposed on faces from the training dataset.
2. To run this code in Matlab you will need to type the command in the Matlab command window. The parameter specification is:

   ```
   draw_pixels(DATA_FILE,INSTANCE_ID)
   ```

   Where `DATA_FILE` is the full path to the location of the `train_faces_clean.mat` on your machine. `INSTANCE_ID` is an integer specifying the instance id corresponding to the face which you wish to display.
3. For example if you saved the `train_faces_clean.mat` file in your home directory and wished to display the face with instance id 7 an example invocation would be:

   ```
   draw_pixels('~/train_faces_clean.mat',7)
   ```

Displaying Eigenfaces

1. The supplied Matlab code `draw_eigenfaces.p` can be used to visualize the top N Eigenfaces from the training dataset.
2. To run this code in Matlab you will need to type the command in the Matlab command window. The parameter specification is:

   ```
   draw_eigenfaces(DATA_FILE,NUMBER_OF_TOP_EIGENFACES)
   ```

Where `DATA_FILE` is the full path to the location of the `train_faces_clean.mat` on your machine. `NUMBER_OF_TOP_EIGENFACES` is a an integer corresponding to the number of top Eigenfaces which you wish to display.

3. For example if you saved the `train_faces_clean.mat` file in your home directory and wished to display the top 20 Eigenfaces an example invocation would be:

```
draw_eigenfaces('~/train_faces_clean.mat',20)
```

You might find it useful to use the *zoom* tool in Matlab to make the displayed Eigenfaces easier to visualize.

Reconstructing Faces using the Eigenface Basis

1. The supplied Matlab code `reconstruct_face.p` can be used to visualize the representation of a face using the N top principal components.
2. To run this code in Matlab you will need to type the command in the Matlab command window. The parameter specification is:

```
reconstruct_face(DATA_FILE,NUMBER_OF_TOP_PRINCIPAL_COMPONENTS)
```

Where `DATA_FILE` is the full path to the location of the `train_faces_clean.mat` on your machine. `NUMBER_OF_TOP_PRINCIPAL_COMPONENTS` is a an integer corresponding to the number of top principal components which you wish to use to reconstruct the face.

3. For example if you saved the `train_faces_clean.mat` file in your home directory and wished to reconstruct the face using the top 100 principal components an example invocation would be:

```
reconstruct_face('~/train_faces_clean.mat',100)
```

Using the WEKA Experimenter

The Experimenter has several advantages: Firstly it provides more information about the results than the Explorer. Secondly it is very convenient to run an experiment in which you compare different datasets, different classifiers, or different parameters for a classifier. Finally, it seems to be more stable than the Explorer.

Below you find brief instructions on how to perform experiments with the simple “Experiment Configuration Mode” in the Experimenter. There is also an advanced mode which can be more convenient (but which is also more complicated to use). You can find more information about the Experimenter on the WEKA website (see lab sheets) or in the Experimenter tutorial: `http://www.scribd.com/doc/2892691/WEKA-ExperimenterT̃utorial`.

If you are unsure whether the Experimenter is doing the right thing, it is easy to check this by reproducing single experiments in the Explorer (the average PC values should be almost identical).

Setting up an Experiment

1. After starting the Experimenter go to the “Setup” tab. Make sure that the “Experiment Configuration Mode” is set to “Simple”.
2. Create a new experiment by clicking the “New” button.
3. Choose the dataset(s) that you would like to run your classifier on: In the “Datasets”-box click “Add new...” and select a data file. The selected data file will then be shown in the list below the button. You can repeat this if you want to add multiple datasets.
4. Choose the experiment type: In the “Experiment Type”-box select “Cross-validation”. Set the number of folds to 5. Make sure “Classification” is selected.
5. In “Iteration Control” set the “Number of repetitions” to 5.
6. Finally, select the classifiers that you would like to train and evaluate on your dataset(s): In the “Algorithms” box click “Add new...”. In the dialog that opens choose the appropriate classifier from the list and set its parameter values. Then hit “OK”. The classifier name and its parameters will be added to the list. Repeat this for all classifiers / classifier parameterizations that you would like to compare.
7. When you are done configuring your experiment you can save the configuration. You can also specify a file to write the results to (but you don’t have to).
8. To start your experiment go to the “Run” tab and hit “Start”.

Inspecting the Results

When the experiment has finished go to the “Analyse” tab.

1. In the “Source” box click on the “Experiment” button to load the results of the experiment that has just finished running. In the “Configure test”box keep the default settings but check the box “Show std. deviations”. If this option is switched on, the standard deviation of the PC across the CV folds and runs will be displayed.
2. Start the analysis by clicking the “Perform test” button. The results will be shown in the “Test output” text field. This text field will contain a table that has one column for each classifier, and one row for each dataset. For each classifier / dataset pair this table contains the average PC (across CV folds and runs) as well as the standard deviation (in parentheses).
3. Under this table there is an extra row containing an entry with a format (../../..). This summarizes the statistical significance of pair-wise comparisons of schemes using the corrected resampled T-Test. The column that has the (v/ /*) format is the baseline for the comparisons. The rest of the columns have an entry with three numbers. (1/0/2) indicates that the scheme of the respective column had 1 significant win, 0 ties and 2 significant losses with regard to the baseline scheme. Additionally the v and * symbols will be displayed inside the table, next to the PC for the row that won or lost respectively.
4. You can configure the presentation of the results in various ways, using the options in the “Configure test” box. E.g. you can choose to see the classifiers as rows and the datasets as columns by clicking on “Select” next to “Row” and choosing “Schemes” in the list that appears and then clicking on “Select” next to “Column” and choosing “Dataset”. This way you can see if any of the datasets has significant wins/losses with regard to a baseline dataset. You can change the baseline by clicking on

"Select base..." next to "Test base" and selecting the scheme that you want to use as baseline. Also, you can change the significance level for the T-Tests by changing the value in the textbox next to "Significance".

5. Each time you change the options in the "Configure test" box, you need to click the "Perform test" button to see the results.