APPLIED MACHINE LEARNING SYSTEMS ELEC0134 20/21 REPORT

SN: 17082653

ABSTRACT

The assignment detailed in this report involves facial detection problems in the context of binary and multi-class classification problems. The approach taken was largely influenced by the results of facial detectors found through literature survey. These were then applied to a provided data-set with 95% feature extraction rate on the celebA data-set, and 81% on the cartoon data-set. Classical machine learning algorithms were used for the classification tasks, with generally good results. In an effort to reduce the complexity associated with the large dataset in task B, an image processing approach was taken to surprisingly good effect, with a simple KNN algorithm giving 93.3% test accuracy in task B2. The unseen test accuracy in task A2 gave lower than expected results, and its possible future areas for improvement lie in data-set analysis and cleaning to ensure training set gives an accurate representation of potential test sets. See footnote to access the code repository. [1]

***Index Terms*—** Machine Learning, Classification, Facial Recognition, Logistic Regression, SVM, KNN

1. INTRODUCTION

This report details the methodologies used in solving two machine learning classification tasks given as a final assessment in module ELEC0134: Applied Machine Learning Systems. Task A is a binary classification problem, whilst Task B is a multi-class classification problem. Both tasks have two distinct sub tasks (A1, A2, and B1, B2). Two data-sets were provided, one for each task set. Task A1 requires the prediction of gender, whilst task A2 has us predict whether a person is smiling or not, both based on an image of a face. Tasks B1 and B2 deal with classifying eye color and face shape into one of five classes, based on an image of a cartoon face.

The models chosen to tackle each task were decided through a combination of consulting relevant literature, data exploration, and preliminary model testing. The literature review summarizes popular facial recognition technique as well as machine learning algorithms for classification. The data exploration section briefly explores the provided data-sets for each task. This is followed by a detailed description of the selected models, and their implementation. After this, the results of the assignment are presented, along with accuracy prediction scores on an unseen separate test data-set. Finally, this is followed by a discussion on the findings, and suggestions for future improvements.

[1]The code is provided at `https://github.com/17082653/AMLS_assignment20_21`

2. LITERATURE SURVEY

In order to be able to supply features from images to machine learning models for training and testing, a suitable feature extraction method is required. Raw image files themselves cannot often be used to good effect. As task A deals with facial detection, long a hot topic in machine learning applications, there exist a wide variety of possible facial detection methods that can be used for feature extraction.

One such method is facial landmark detection. Given an input image, a facial landmark detector attempts to localize a set of key facial features on a face, like the mouth, eyes, nose, and jaw. A 2018 journal by Johnston and Chazal provides a great review of most cutting edge facial landmark detection techniques [1]. Their review includes an overview of publicly available facial landmarking databases, as well as benchmark results for 'off the shelf' frontal face detectors. The benchmark results found that the dlib HOG-based (histogram of oriented gradients) face detector outperformed all OpenCV face detector variants. Dlib is a C++ based machine learning toolkit library (with a Python wrapper), used widely in industry and academia [2], whilst OpenCV is perhaps even more well known as a popular computer vision library [3]. Because of this undeniably great performance, dlib's HOG-based face detector was chosen for face detection, and dlib's pre-trained facial landmark detector based on an implementation from a 2014 paper "One Millisecond Face Alignment with an Ensemble of Regression Trees" [4] was chosen for landmark detection. This pre-trained landmark detector has been trained on the "iBUG 300-W data-set" [5]. This data-set uses a 68-point markup to annotate all of its images. This means the pre-trained dlib detector used in this assignmenet is trained to extract 68 points from an image of a face. This will be expanded upon in section 5. A general review of supervised ML algorithms is available at [6]. This was used to get a detailed comparison of popular learning algorithms, and their strengths and weaknesses.

3. DATA EXPLORATION

This section aims to give a brief insight into the provided data-sets. Figure 1 provides two example images from the data-sets.

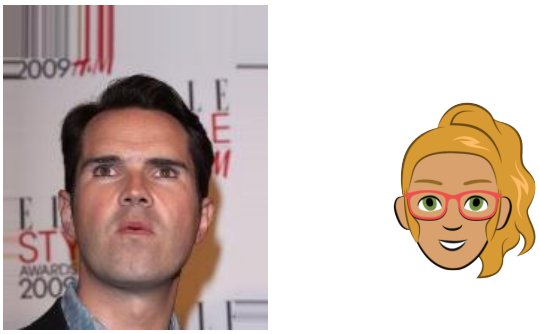

Fig. 1. CelebA sample (left), Cartoon Set sample (right)

3.1. Task A: CelebA Data-set

Task A utilizes a sub-set of the CelebFaces Attributes Data-set (CelebA), which contains 5000 jpg images of celebrity faces [7]. Each image is 178 × 218 pixels, and images have been digitally manipulated to position all faces at the center. Provided with the images was a csv file, were each image is additionally labeled: -1 or 1 for female or male, and 1 or -1 for smiling or not smiling respectively.

3.2. Task B: Cartoon Data-set

Task B utilizes a subset of "Cartoon Set", which contains 10,000 png images of cartoon faces, provided by Google [8]. Each image is 500 x 500 pixels, and all images were additionally labeled with: their eye color, and their face shape. There is a total of five possible eye colors and five possible face shapes.

4. DESCRIPTION OF MODELS

4.1. Task A: Logistic Regression

Having conducted the initial test for the eight basic models, the polynomial kernel SVC and logistic regressor were chosen for further exploration and testing. After some tests (which are explained further in section 5), the logistic regressor was chosen as the model of choice for the task.

Logistic regression is a statistical model not too dissimilar to linear regression. The method attempts to predict the outcome for a binary variable, from one or several input variables. The output can be a categorical variable, or a continuous one, which distinguishes it from linear regression. This makes the method better suited for binary classification as its output always lies between 0 or 1, whereas a linear regression classifier can have an output less than 0 or more than 1. The hypothesis of the model works like so:

$$h_\theta(\boldsymbol{x}) = g(\boldsymbol{\theta}^t \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^t \boldsymbol{x}}} \quad (1)$$

Where $\boldsymbol{\theta}$ represents the parameter vector, and $\boldsymbol{x}$ the feature vector. The logistic or Sigmoid function can then be visualized like so (see figure 2):

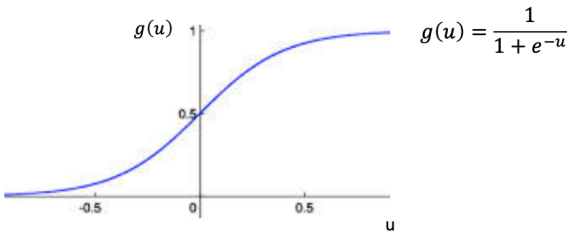

Fig. 2. A logistic function

The decision regions for such a classifier would then lie above and below 0.5. That is, if after supplying the input feature vector, the predicted output is $\hat{y} < 0.5$, it can be classified as 0, if it is $\hat{y} \geq 0.5$, it is classified as 1.

Since both task A1 and A2 share are problems of binary classification, given a good feature set, this relatively simple model ended up performing just as well as more complex ones during testing, which is elaborated on in section 5. The big upside of choosing this model over an SVC, is the reduced complexity and therefore training time. Tuning hyper-parameters and training the model can be done in a fraction of the time. In fact, the final training time for task A1 and A2 was just 1.68 and 1.78 seconds respectively. The model was chosen for both task A1 and A2.

4.2. Task B1: Support Vector Classifier

An SVM (support vector machine) is a model which attempts to algorithmically build an optimal separating boundary (known as a hyperplane) between data-sets by solving what is a constrained quadratic optimization problem [9]. The idea is to separate data with a classification boundary that also has some 'margin'. In the case of linearly separable data, one can apply what is known as a linear SVM. Figure 3 shows what a classification margin for linearly separable data-set might look like. [10].

The three samples closest to the hyperplane are the support vectors. The hyperplane boundary is decided based upon these samples, as these are the data points the margin pushes up against. The optimization problem comes in the form of solving for the decision boundary so that the data of both classes is as far from the margin as possible. One of the parameters to be tuned in SVC's is the regularization parameter, which effectively dictates the tolerance for mis-classifications. Lowering this value allows one to tune

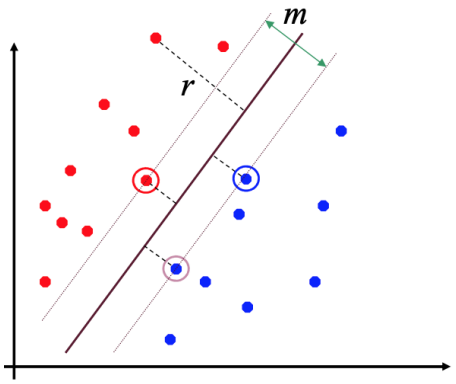

Fig. 3. The classification margin of a linear SVM

the hyperplane boundary in a way that prevents over fitting. Additionally, by using what is known as a kernel function, SVM's can be extended to include varying degrees of non-linearity and flexibility [9]. The simplest one of these is the linear kernel, however a polynomial or radial basis function (rbf) kernel can be used instead to create different 'shapes' of classification boundaries. One can easily imagine how the boundary in figure 3 might look like if a polynomial kernel of degree 3 or 5 were to be applied. This flexibility with kernel functions makes the SVM a good choice for a multi class classification problem like the one in task B1.

4.3. Task B2: K-Nearest Neighbors

Though initially the application of an SVC to task B2 also seemed like a reasonable idea, the final feature extraction process turned the data-set into one not at all similar to those in tasks A1, A2 and B1 (elaborated on in section 5). The reduced size of the final feature set allowed for the use of a K-nearest neighbors classifier (KNN). The KNN algorithm is one of the simplest machine learning algorithms. This is because in contrast to other algorithms which utilize a training set to extract a hypothesis which can be used for new predictions, KNN algorithms utilize the entire data-set to make predictions for new test points. The algorithm works by classifying a data point based on the classes of k-number of known data points closest to the unknown data point. The algorithm used to compute the closest neighbors can vary, however there will always some metric which returns the 'distance' between two points [11].

5. IMPLEMENTATION OF MODELS

5.1. Task A: Data-set Pre-processing

5.1.1. Dlib library and 'utility.py'

As mentioned in the literature review, dlib's pre-trained facial landmark detector was decided on as the method for feature extraction. When an image is passed to this detector, it attempts to localize 68 landmarks on the face. Figure 4 shows the locations of these landmarks. The 'utility.py' folder contains the functions required to make this happen. The image data-set directory is iterated through in the 'extract_feature_labels' function, each image loaded to an array through a function from the Keras library. The image is then passed to the 'run_dlib_shape' function, where it is converted to gray scale using OpenCV. This conversion is generally recommended in order to reduce the potential for noise, as well as the size of the image. The gray scale image is then passed into the dlib face detector. Dlib's detector highlights a rectangular region of interest (ROI) within which a face was located. This rectangular ROI is then passed to the pre-trained facial landmark predictor. Within this region, the detector looks for a face and localizes the landmarks. Figure 5 demonstrates the output of this process.

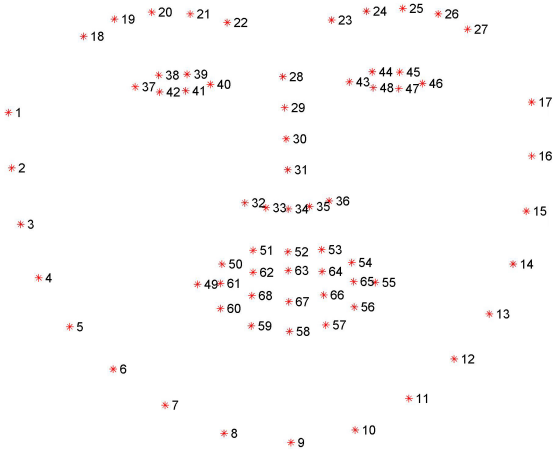

Fig. 4. 68 facial landmark coordinates from the iBUG 300-W dataset, dlib pre-trained detector

The 'pre_processing.py' file deals with reshaping the dataset into correct dimensions. At the end of the process, we are left with the extracted image landmarks, and their corresponding labels from the data-set (gender label, or smiling-or-not label). On average, it was found this method extracted landmark features from 95% of the images in the data-set.

5.2. Task A1: Gender Detection (Male or Female)

5.2.1. Scikit-learn, 'pre-processing.py' and 'models.py'

Having extracted the features and processed the data, one can begin to work on classification. To begin, eight different base ML models were trained and tested on the data with all features included, in order to evaluate initial accuracy. The function used for this can be found in the 'models.py' file. Each model tested was of the basic implementation provided by the scikit-learn library. This is a popular classical machine learning library, with a wide variety of implemented algorithms and tools [12]. The data was split using an 80/20 split for

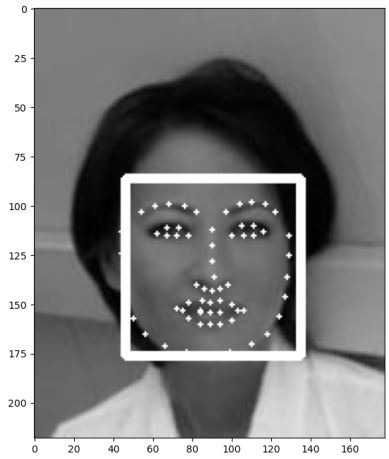

Fig. 5. Sample face with highlighted ROI and landmarks

training and testing respectively. The data was additionally shuffled before the split. This improves the changes of achieving a well proportioned split in terms of the types of images in both the training and test data. This was done through a scikit-learn 'train_test_split()' function, and can be found in 'pre_processing.py'. It is important to note that validation data has not been split off the training data yet, as model validation is later done using scikit-learn k-fold cross validation routines.

Model	Test Accuracy
KNN	78.7
SVC (linear kernel)	85.1
LinearSVC (liblinear implementation)	90.5
SVC (polynomial kernel)	91.4
Logistic Regression	88.3
Gaussian NB	67.3
Random Forest	86.2
Gradient Boosting	88.1
Average	84.5

Table 1. Initial model test results

Table 1 shows that the results across the board were quit good. This can most likely be attributed to the great perfor mance of the landmark detector and feature extractor. Th best performing models were: a logistic regressor (88.3%) a support vector classifier (SVC) with a polynomial kerne (91.4%), and an SVC with a linear kernel (90.5%) based o an implementation of the liblinear library. This in contrast t the normal SVC in scikit-learn, which is based on the libsvm library. These initial tests helped to confirm the background reading. with SVC's performing well, as expected, and logistic regression in second.

5.2.2. Feature selection and 'validation.py'

With the initial test done, feature selection was undertaken in order to decided on the best features to train and test the data on. For this purpose, the 'feature_selection()' function was created. This function takes in the training and test data, an array of score functions, and an array of score threshold values. Two score functions were used for evaluating features: 'chi2' and 'f_classif'.

The chi-squared test is a uni-variate feature selection method. The test measure dependence between stochastic variables. This means it is able to filter out low scoring variables which are likely independent of class, and therefore irrelevant for classification [13].

The 'f_classif' score function computes the ANOVA (analysis of variance) F-value for a provided sample. The F-statistic is the ratio of two variances. Computing within-group variance between variables allows one to determine the amount of random error in the data. Keeping this low will generally result in better accuracy [14].

The 'feature_selection()' function iterates through these two score functions, as well as an array of 7 score threshold values. The scikit-learn 'SelectKBest()' is used with each of these to find the k-number of features which fall above the thresholds. K is set to 'all' however, and custom score thresholds are used for determining the number of features. All eight basic models are then trained and tested on just these features. The features correspond to the 68-facial landmark points.

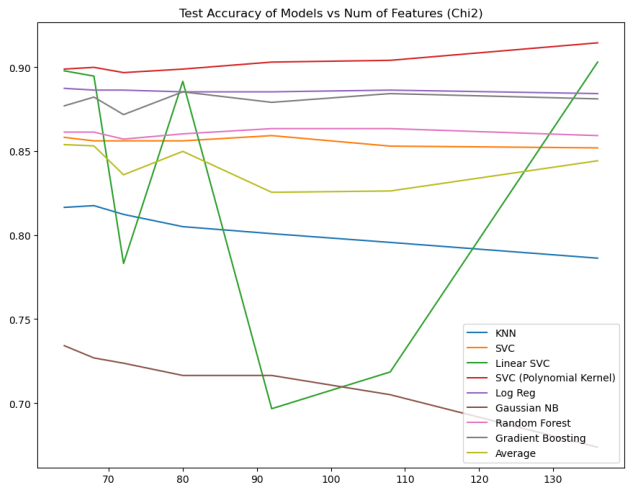

Fig. 6. Accuracy of models vs. number of features selected using the chi2 score function

As figure 6 shows, running the chi2 feature selection algorithm on the models did not result in uniform accuracy increase across models. In general, the polynomial SVC performed best with all features included, while other algorithms performed better with a selected feature set. This means that the best performing algorithm was still the polynomial SVC with all features. The ANOVA F-value based feature selection gave similar results.

5.2.3. Parameter tuning and 'Grid_SearchCV()'

While feature selection alone did not substantially increase accuracy in this task, the combination of feature selection and parameter tuning did give good results. The scikit-learn 'Grid_SearchCV()' function performs an exhaustive search of the best selection of given parameters and their values. The method implements a fit and scoring method, and uses 5-fold cross validation on the passed training data-set to determine the best parameter combination. This method was ran on the polynomial SVC, and the logistic regression classifier.

The optimal parameters for the polynomial SVC were found to be C = 0.1, and degree = 4. C is the regularization parameter, while the degree is simply the degree of the polynomial used to for classification. The parameters for the logistic regression classifier were found to be C = 0.1 and solver = 'newton-cg'. The solver parameter specifies the type of algorithm to be used in the optimization problem. Each one performs slightly differently dependant on the given data-set, and so optimizing for it can give good increases in performance.

Having tuned the parameters, the feature selection algorithm was ran on these two classifiers. Figure 7 shows the results of this. The ANOVA feature selection proved much better than chi2 in this case. The tuned logisitc regression classifier was seen to outperform the SVC, especially with an optimal selected feature amount of around 114/136.

5.3. Task A2: Emotion Detection (Smiling or Not Smiling)

Task A2 followed the same method for model selection and parameter tuning as task A1. Since the task deals with Once again, a logistic regression classifier ended up being used for classification.

5.4. Task B: Data-set Pre-processing

Task B utilized the approach from A for B1, and a different approach for feature extraction for B2. As task B1 has us classify five different types of face shape, (mostly identifiable by the jawline of the cartoon) it was thought that applying the face landmark detector to the cartoon data-set could give similar results as in task A.

Applying this same approach to the cartoon set results in feature extraction from around 81% of the images in the data-

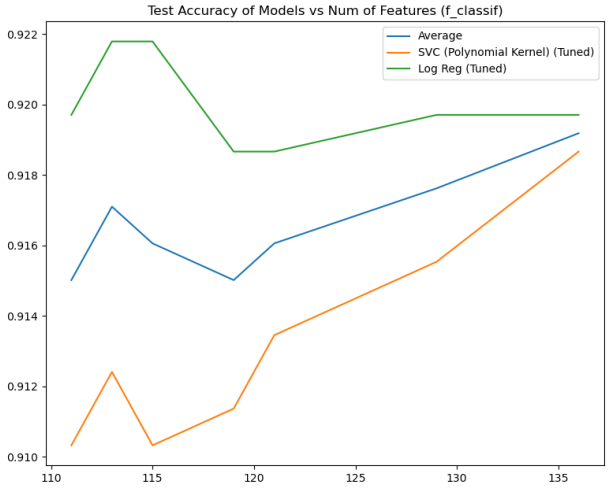

Fig. 7. Accuracy of tuned models vs. number of features with f_classif feature selection

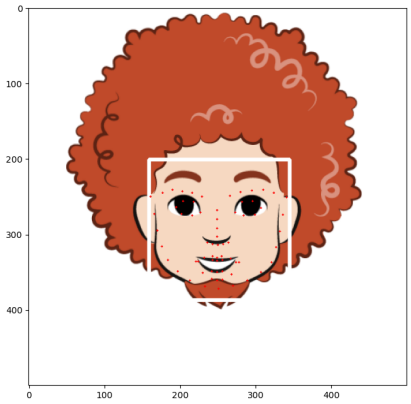

Fig. 8. Sample cartoon with highlighted ROI and landmarks

set. The drop in the feature extraction rate can be largely attributed to the fact the detector has not been trained on a cartoon data-set, and so it struggles with facial recognition. Additionally, even though landmarks were extracted relatively often, as figure 8 demonstrates, they often fell outside of/did not align with the appropriate regions. This had a noticeable impact on classification accuracy (elaborated on in section 6).

5.4.1. OpenCV and eye_extractor()

The approach adopted for task B2 was different. As the task has us classify the eye color of the image, facial landmarks alone would not suffice. However, the fact that the cartoon

data-set is very uniform and clean, means that facial features on each image are always located in the same region. This allowed the use of the OpenCV computer vision library to crop every image around the eye region. Having cropped the image to one of the eyes, a mask is then applied to the image. Each of the five eye colors in the data-set (blue, brown, green, gray, black) was sampled with a color picker for its RGB (red, green, blue) value. Based on this value, a set of lower and upper color boundaries was created for each. The mask works by using the OpenCV 'bitwise_and()' function with the cropped image and the mask. This leaves behind only the colors within the range specified by the boundary (see figure 9). The type of mask applied corresponds to the eye label of the cropped image passed into the function. The 'eye_extractor()' function in 'utilityB.py' details this process.

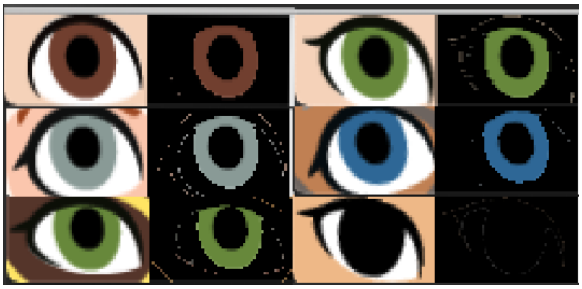

Fig. 9. Side-by-side comparison of cropped image, and the image with a successful mask applied

After the mask is applied, an OpenCV 'mean()' function is used to compute the mean of each color channel. This gives a list with the blue, green, red mean value of the image (eg. (13.4, 27.2, 22.3)). This list is returned from the 'eye_extractor()' to be used as features. By the end of this process, each image has three features in the form of a red, green, blue value. This feature extraction process works relatively quickly, even on what is a relatively large data-set. It extracts the features from 100% of the images, however some of these are bogus values, as some cartoon images have sunglasses which obstruct the color of the eye.

5.5. Task B1: 5 Types of Face Shapes

Since this task is essentially a multi-class version of the facial landmark detection problem from task A, the SVC and logistic regression classifier once again performed best. After performing a model test with the tuned models, the polynomial SVC ended up performing better in this problem.

5.6. Task B2: 5 Types of Eye Color

Having extracted the BGR color features from the dataset, the task left is a simple case of multiclass classification. For this, the base model test was ran once again. The best performing model was a base scikit-learn implementation of the KNN (K-Nearest Neighbors) classifier, which ended up being used as the final classifier for the task. The default number for k was used, which was 5.

6. EXPERIMENTAL RESULTS AND ANALYSIS

Table 2 below, lists the final results of all models for all tasks. The training accuracy is the accuracy achieved by a model on its train/test split. The validation accuracy is a 5-fold cross validated accuracy score ran on the training data, and printed in 'main.py'. The test accuracy is the accuracy of the models on an unseen data-set provided by the module organizers, not used in the training and validation process. As can be seen all models performed relatively well across the board. The notable exception being the test accuracy of the logistic regression classifier model in task A2. While the training and validation score are of appropriate value, the test accuracy is quite low, at 43.2%.

Task	Model	Train Acc	Val Acc	Test Acc
A1	Log Reg.	92.2	91.8	91.2
A2	Log Reg.	90.5	89.0	43.2
B1	SVM (Poly)	73.4	72.5	73.8
B2	KNN	93.1	91.6	93.3

Table 2. Final results of all models

7. CONCLUSION

To conclude, a total of 8 types of classical machine learning algorithms were tested on provided data-sets. For tasks A1, A2, and B1, feature extraction was performed utilizing dlib's pre-trained facial and facial landmark detector. Based on the initial tests, the algorithms to continue exploring were narrowed down to a logistic regression classifier, and a support vector classifier.

Due to the binary classification nature of problems A1 and A2, the logistic regression classifier was used. In both tasks, it performed quite well, with validation accuracy's of 91.8% and 89.0% for tasks A1 and A2 respectively. However, it received a surprisingly poor test accuracy on an unseen dataset in task A2. To fix this in the future, care should be taken to look over the test data-set, and make sure proper feature extraction took place. It is possible the the low accuracy is a result of a mix up in data pre-processing and image labeling. Another way to alleviate this error might be to look into different training/testing splits to ensure any bias in the training data-set when compared to the unseen test set is negligible.

For task B2, a different approach was taken, which involved heavy use of the OpenCV computer vision library. Images were cropped, and segmented by colors by applying a

mask to the image. The resulting blue, green and red mean values of the image colors were used as the feature set for training and testing. This proved to work surprisingly well, with the reduced feature set allowing the use of a KNN algorithm which could leverage the whole data-set. The final validation accuracy for this task was 91.6%, with a test accuracy of 93.3%.

8. REFERENCES

[1] Benjamin Johnston and Philip de Chazal, "A review of image-based automatic facial landmark identification techniques," *EURASIP Journal on Image and Video Processing*, 2018.

[2] "dlib.net," `http://dlib.net/`.

[3] "Opencv," `https://opencv.org/about/`.

[4] Vahid Kazemi and Josephine Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *IEEE Conference on Computer Vision and Pattern Recognition*. KTH, Royal Institute of Technology, 2014.

[5] G Tzimiropoulos S. Zafeiriou M. Pantic C. Sagonas, E. Antonakos, "300 faces in-the-wild challenge: Database and results," in *Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild"*, 2016.

[6] Awodele O. Hinmikaiye J. O. Olakanmi O. Akinjobi J. Osisanwo F.Y., Akinsola J.E.T, "Supervised machine learning algorithms: Classification and comparison," in *International Journal of Computer Trends and Technology (IJCTT)*, 2017, vol. 48.

[7] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[8] Google, "Cartoon set," `https://google.github.io/cartoonset/`.

[9] Stephan Dreiseitla and Lucila Ohno-Machadob, "Logistic regression and artificial neural network classification models: a methodology review," in *Journal of Biomedial Informatic*, february.

[10] Yiannis Andreopoulos, "Online regression statistical assessment of performance support vector machines," AMLS week 6 lecture.

[11] Miguel Rodrigues, "K-nearest neighbors," AMLS week 10 lecture.

[12] "Scikit-learn," `https://scikit-learn.org/stable/`.

[13] "Scikit-learn: Chi-squared feature selection," `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2`.

[14] Jim Frost, "Statistics by jim," https://statisticsbyjim.com/anova/f-tests-anova/.