Fabian Chan, Jiayi Wang
Kaggle Team Name: Green Tea

CSC411 Assignment 4: Facial Expression Prediction

Introduction

We have identified 3 aspects of the system that are worth exploring to obtain a higher classification rate: improving the training set via preprocessing, improving the classifier algorithm, and adopting an ensemble approach using other classifiers.

A dataset of facial expressions contains a lot of variance, due to the varied appearances a person can take on and how the picture is taken. Therefore a method to deal with this variance is needed. To that end, we generated more training data by shifting the images in different directions (to introduce the idea of translational invariance) and to use edge features rather than intensity features (to introduce invariance in terms of skin color and camera lighting).

Gaussian smoothing and Principal Component Analysis were also investigated as methods of removing noise and features irrelevant to facial expressions.

A series of classifiers was tested to identify algorithms with the best performance. k-Nearest Neighbors (k-NN), Support Vector Machine (SVM), Decision Tree, and Logistic Regression were tested using scikit-learn's (a public library) implementations of the algorithms. Bagged versions of k-NN and SVM were also tested along with Ada-boosted Decision Stumps and a Random Forest. Furthermore, 8-fold cross validation was used to determine the optimal hyperparameters for each algorithm. The goal was to investigate a range of different classifiers to be used in combination. To combine the predictions of different algorithms, we tried a simple majority voting strategy. This was done under the assumption that the algorithms we used are sufficiently independent in their approaches that each result is also independent.

Another approach we tried was clustering the training images using k-means, and training a classifier for each of the clusters. The idea is that the images can be grouped into clusters of male and female faces, and perhaps also into clusters of Caucasian, Black, and Asian faces etc. A classifier that is good for classifying expressions within a certain cluster can then be trained. For example, a classifier trained on male faces will be good at predicting male expressions. This was essentially another attempt to reduce variances irrelevant to facial expressions.

Our Submitted Solution

Our submission consists of an algorithm with the following description:

The algorithm is split into two steps: preprocessing and training. In the preprocessing step, four more images are generated from each training image by a shift in each of four cardinal directions. These new images are marked with the label of the original image and added to the training set. Each image in the training set is processed using a Gaussian Laplace filter. Finally, each image is normalized to have zero mean and unit variance. The point of the preprocessing step is to diminish variances that are irrelevant to facial expressions.

During training, an SVM classifier implemented in scikit-learn is used. A one vs one scheme is used to extend SVM to handle multiple classes. Using N-Fold cross validation, the best kernel to use was a "radial based function kernel" with capacity constant of 1.6 (C=1.6, kernel = 'rbf').

Results and Discussion

We systematically evaluated the k-NN classifier over different values of k, and selected the best k based on classification rates obtained through 8-fold cross validation. Figure 1 shows k-NN classification rates for various values of k. The best k for the k-NN algorithm was found to be 7 with a corresponding rate of 61.95%, based on our Python implementation.

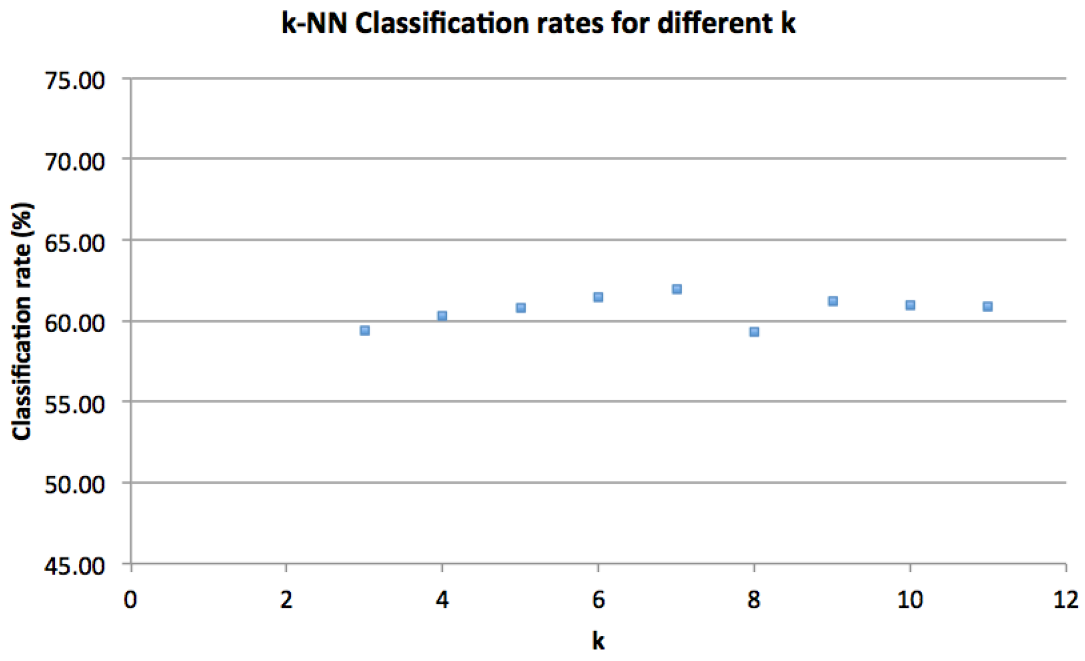

Figure 1. Validation rates for k-NN with different values of k

As an alternative to k-NN, we used the Gaussian Naive Bayes classifier. We wanted to investigate how much the interdependence of the pixel values matter in terms of expression detection by checking the performance of an algorithm that assumes no such interdependence. A lower classification rate of 58.12% suggests that assuming the pixels are independent and identically distributed is a bad assumption. In fact, we fully expected this result by reasoning that each edge in an image is defined by a set of neighboring pixels. Perhaps taking advantage of this fact in our solution would yield a classification performance that surpasses both of the approaches mentioned above.

We believed that an SVM classifier would perform better than k-NN, since k-NN is more susceptible to class noise. k-NN also relies on a distance measure which becomes less meaningful in very high dimensions (note that the images are in 1024 dimensional space). In addition, multiclass k-NN for a large number of classes will often encounter ties during voting, and more frequently so if k is low. However, if k is increased to address this issue, the decision boundaries will become less flexible. In contrast, SVM still provides a nonlinear decision boundary without the costs mentioned above.

Therefore, we focused our efforts on tuning an SVM. The hyperparameters for the classifier was determined using the same cross validation method. The best capacity constant C was determined to be 1.6, while the best kernel function to use was found to be either a radial based function (rbf) kernel or a 3rd degree polynomial kernel. Figure 2 and Table 1 show our cross validation results for different hyperparameters. Using these settings, the classifier performs much better than both GNB and k-NN: resulting in a rate of 74.05% for the rbf kernel, and 74.00% for the 3rd degree polynomial kernel. This increase in performance was expected: an SVM does not assume that the features are iid and its sensitivity to class noise can be easily tuned by a regularization parameter, therefore it should perform best out of the three.

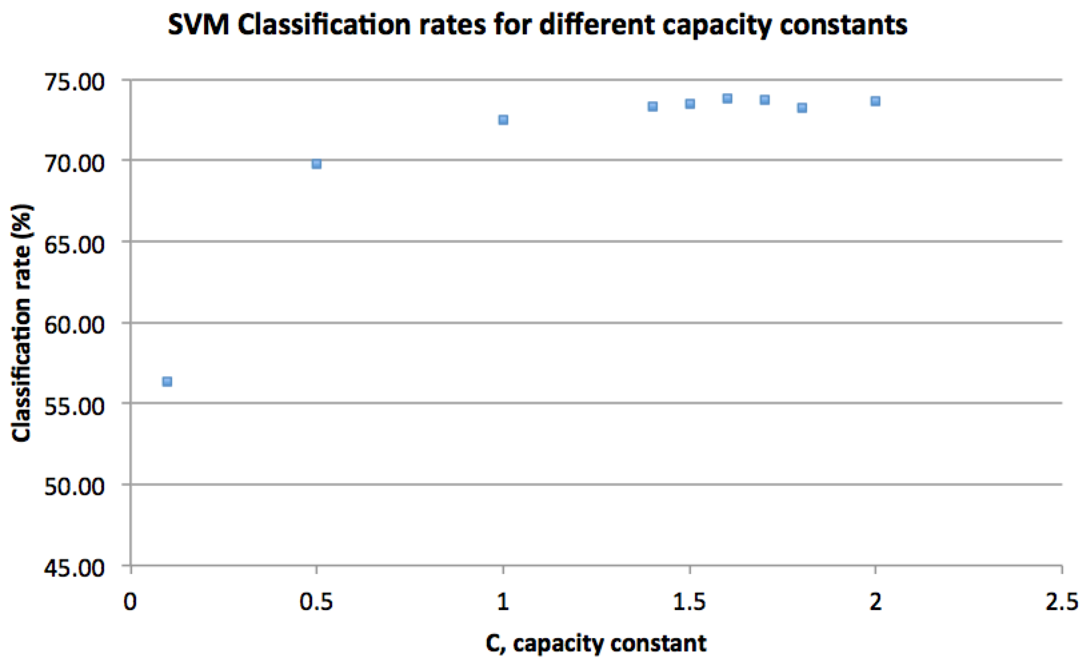

Figure 2. Validation rates for an SVM with different capacity constants

Table 1: Validation rates for SVMs with different kernels (capacity constant set to 1.6)

Kernel	Linear	Poly 2nd	Poly 3rd	Poly 4th	Poly 5th	Poly 6th	RBF	Sigmoid
Rate	69.61	73.57	74.00	73.11	72.17	70.56	74.05	29.47

While exploring some alternatives to k-NN, we attempted some ensemble approaches such as a Random Forest and bagged k-NN. However, these methods were quickly discarded as we could not tune the parameters to perform better than 67%. One reason why bagging did not worked well is that there is not enough variety in the data. The classifiers trained through bagging are not sufficiently different from each other. Particularly, in the case of bagged k-NN, the explanation is that bagging does not work well with stable classifiers. If minor perturbation in the input data does not cause a change in the model, the classifiers trained through bagging would not perform sufficiently differently. Since k-NN is a stable classifier, bagging k-NN would not improve it. [1]

To improve the performance of our classifier, we decided to generate more training data by shifting the original image by one pixel in each direction as an attempt to account for the translational invariance of expressions. With 4 times more training data, however, the training time of our classifier increased from about 30 seconds to 380 seconds. The performance of the SVM increased to 74.39% and 75.05% for the 3rd degree polynomial kernel and the rbf kernel respectively. We decided to continue our investigation with only the rbf kernel because of its larger increase in performance.

A further step we took to enhance our classifier was to use edge features instead of intensity features of the images. The idea is that the expression of a face is invariant to the intensity of the pixels (i.e. people with lighter skin will have overall lower intensity values than people with darker skin color, but they can both have the same expression). Therefor we preprocessed both the training and test images with the Gaussian Laplace filter as an extra step in our current pipeline. The performance of the algorithm is further improved to 77.11%.

Finally, we made a deliberate decision to not use any unlabeled images in our final algorithm. During our investigations, we attempted to cluster images into groups and train a classifier for each cluster. The idea was that samples with features irrelevant to facial expressions would be clustered together if they are of a similar type. For example, there could be clusters based on gender and/or race. Within these clusters, the algorithm will be able to better pick up on variances that do contribute to expression detection. However, this approach did not yield good results. A deeper inspection of the clustering process revealed that many clusters were dominated by a subset of the classes, creating a bias in the data that makes training a well-performing classifier difficult.

Conclusion

The classification rate of our algorithm on the public test set turned out to be 77.75%, which is reasonably close to our validation score. Out of all the approaches we explored, what improved our results the most were preprocessing methods that reduce variances irrelevant to facial expressions. This makes sense, as there are a lot of significant variations in facial features from person to person that is not relevant in terms of expression detection (e.g. skin color, facial hair). Investigation of better features that are invariant to such irrelevant features is something we would consider to further improve classifier performance.

References

[1] Breiman, L. (1996a). Bagging predictors. Machine Learning 26(2), 123–140.

Instructions to Run Code

1. Install Python 2.7 from https://www.python.org/downloads/

2. Install Scikit-learn, a free machine-learning module for python. Follow the instructions on http://scikit-learn.org/stable/install.html

3. Install Scikit-image, a free image-processing module for python. Follow the instructions on http://scikit-image.org/download.html

4. Move public_test_images.mat, labeled_images.mat, hidden_test_images.mat, main.py, and run_model.py into the same directory.

5. Run main.py. This will generate predictions.csv containing the predictions. and a trainedModel.pkl file that contains the trained model; it should take about 5 minutes to run.

6. Run run_model.py. This will load the trainedModel.pkl generated from the previous step and will use it to generate predictions in the file test.mat, as required.

Due to the size of the trained model file, trainedModel.pkl (228MB), it cannot be uploaded to MarkUs. Instead, please generate it by simply running main.py in step 5.