CSC411H1 Winter 2016

Assignment 3 Write-Up

Mingye Wang, Kaggle Username: WangL21, CDF ID: g3nicky
Sijia Wang, Kaggle Username: WangPeter, CDF ID: g2peter

Introduction

In this assignment, various techniques including k-Nearest Neighbour, Neural Networks and Linear classification will be applied in order to develop a highly reliable machine learning model capable of achieving over 80% accuracy in classifying 2,925 labeled and 98,058 unlabelled images into 7 facial categories.

After a first glance at the problem and training data provided, we couldn't decide right away which method to use in order to achieve best results.

We thought of using a multi-layered neural network, coupled with convolutional layer, to build the model. However, after carefully examining training data and reviewing lecture slides about Ensemble Methods. We came to the conclusion that building model by a single machine learning method would not do very well because each method, no matter if it is a convoluted neural network, a K-nearest neighbour, or a multi-variant logistic classification, they all have their weaknesses. In addition, with the training set being quite large, it would probably be difficult to fine-tune a single method to perfection in the span of a few weeks.

Instead, we decided to experiment on training multiple models using various techniques hoping by combining their strengths we could get a better result over using any single one of these methods alone.

In this assignment, we eventually used combination of k-nearest neighbour (different from the provided baseline), logistic classification, neural network, and support vector machines (SVM) to achieve our best classification score of 79.7%.

Solution Description

Feature Extraction

For this assignment, we used Gabor Filters for image feature extraction. The biggest advantage of Gabor filters is its robustness under noises such as illumination and variant in image itself such as rotation, scale and translation. (Kamarainen et al., 2006) Using the MATLAB script publicly available online (see reference for detail), we were

able to extract features from images into a M x N matrix, where N is the number of training samples and M is the number of Gabor features extracted. For this assignment, we chose 2560 Gabor features per image.

Training with Different Classifiers

K-Nearest Neighbours

We initialized our K-Nearest Neighbours model using MATLAB's built-in function templateKNN(). With multiple experiments on training data using cross validation and reference from provided sample code, we eventually chose the number of neighbours to be 5 for best results.

Afterwards, we used publicly available MATLAB built-in function fitcecoc to fit training data to labels provided. We set the 'CorssVal' option to 'on' to prevent overfitting.

Last, we used MATLAB built-in function kfoldLoss on the trained and cross-validated model to print out cross-validation error rate in order to get a better understanding of how the model is doing. For cross-validation, we used the default 10-fold cross validation for better balance between efficiency and performance. We made use of this error rate to help us select the most suitable number of neighbours for this training set.

Support Vector Machine (SVM)

We initialized the support vector machine model using MATLAB's built-in function templateSVM(). After experimenting with two different kernel functions, Gaussian kernel function and Linear kernel function, we chose linear kernel function over Gaussian for better performance because training with linear kernel function was much faster than that with a Gaussian kernel function yet performance of the two functions were similar.

Then, we used publicly available MATLAB built-in function 'fitcecoc' to fit training data to labels provided. We set the 'CorssVal' option to 'on' to prevent overfitting.

Lastly, we used MATLAB built-in function 'kfoldLoss' on the trained and cross-validated model to print out cross-validation error rate in order to get a better understanding of how the model is doing. For cross-validation, we used the default 10-fold cross validation for better balance between efficiency and performance. We made use of this error rate to help us select the better cost-effective kernel function between linear and Gaussian kernel functions.

neural Network

We initialized the neural network model using MATLAB's built-in function patternnet(). After researching the method carefully, we used its default training function 'trainscg' because this conjugate gradient method provides a very effective way to optimize large, deterministic systems by gradient descent(Nicol N. Schraudolph & Thore Graepel,

2002). We set the number of hidden units to 60 for better fitting.

During initialization we used 75% of the data for training and 25% of the data for validation in order to prevent overfitting, this also gave us a general idea of who the network performed with different number of hidden units.

We used publicly available MATLAB built-in function ‘train’ to train the neural network with training data and corresponding labels.

Linear classification

We initialized the multivariate linear classification model using MATLAB’s built-in function templateLinear(). We chose support vector machine (SVM) as the linear classification model type after experimenting on both logistic function and support vector machine during which support vector machine gave a much better results on classification errors.

Then, as previously applied on other models, we used publicly available MATLAB built-in function ‘fitcecoc’ to fit training data to labels provided. We set the ‘CorssVal’ option to ‘on’ to prevent overfitting.

Lastly, we used MATLAB built-in function ‘kfoldLoss’ on the trained and cross-validated model to print out cross-validation error rate in order to get a better understanding of how the model is doing. For cross-validation, we used the default 10-fold cross validation for better balance between efficiency and performance. We made use of this error rate to help us select the better classification model between logistic function and SVM.

Combining Different Models

After all 4 models finished training and produced each of their own predictions for the test data, we proceeded to combine those results for the prediction.

To obtain the most confident result, we wanted to choose the majority vote from classification result of all four models above. This approach is similar to the K-nearest neighbour (KNN) method, which also select the majority vote from the training sample’s k nearest neighbour labels.

In the case of a tie, we chose to use the result of support vector machines over others because it almost always performed best among the 4 classifiers on cross-validation during training phase regardless of different parameters we have tried during the training phase.

Trails and Errors

During the course of pursuing a higher classification rate for the given testing data, there were several moments that we were stuck and didn't know what had gong wrong and only by experimenting with different combinations of parameters and classifier models were we able to achieve a better result.

Parameter Setting for Individual Classifiers

Because our model's final decision is fairly simple (talking the majority vote of multiple candidate classifiers), candidate classifiers is independent of each other. Therefore, it is reasonable to fine-tune each of the classifiers individually to achieve the overall best performance.

Neural Network

We experimented total of 4 set of hidden units; 10, 30, 50 and 60 hidden units. Because each training took quite a long time, we did not continue to enlarge the number of hidden units. The loss function of the hidden units was set to cross entropy.

10: Classification accuracy on validation set — 70.212%
30: Classification accuracy on validation set — 70.8921%
50: Classification accuracy on validation set — 72.5323%
60: Classification accuracy on validation set — 73.7%

K-nearest neighbour

We experimented 4 values for K-nearest neighbours; K = 3, K = 5, K = 7 and K = 9. Before we continued to increase value of K, we noticed there was an increase in loss value of 10-fold cross validation from K = 5 to K = 7 in validation set. This indicated an overfitting possibility so we decided to fix the K value to 5.
K = 3: Classification accuracy on validation set — 57.1295%
K = 5: Classification accuracy on validation set — 62.1224%
K = 7: Classification accuracy on validation set — 60.1095%
K = 9: Classification accuracy on validation set — 59.0528%

linear classification

We experimented with both logistic linear classification function and support vector machine (SVM) for linear classification model with initial bias of 0, learning rate of 1/(training size) and L2-regularization.

The result shows a better fitting from support vector machine function. This is probably because of support vector machine is better at separating data with high dimensions because it only considers data points near the margin while logistic function considers all data including some noise.

Also, because MLE is calculated at every dimension, logistic regressions could get some worse results near the margin area compared to support vector machines.

For this model, there was still much to be experimented on; type of regularization method, value of bias, learning rate, etc. but unfortunately we ran out of time.

SVM: Classification accuracy on validation set — 79.2623%
Logistic: Classification accuracy on validation set — 78.2903%

support vector machine
We experimented on both linear (sigmoid) and Gaussian kernel function for support vector machine.

Linear: Classification accuracy on validation set — 79.66%
Gaussian: Classification accuracy on validation set — 72.2045%

Removal of Naive Bayes Classifier

We initialized our NaiveBayes model using MATLAB's built-in function templateNaiveBayes() with normal distribution as its distribution.

For Naive Bayes classifier, there was not much to experiment on, we simply set the distribution of the model to normal distribution.

However, we doubt if Naive Bayes classifier could help much because its strong assumption of feature independence. As we can see form the following results, using Naive Bayes classifier results in poor classification rates which indicates that its assumption of features being independent from each other is not true in this case.

Classification Rate using only Naive Bayes — 59.599%

Conclusion

Out classification model eventually achieved 77.751% of classification accuracy on the public data set and 79.7% of classification accuracy on the hidden data set.

We were a bit disappointed seeing this result at first. However, after reviewing our methods and model design, we concluded that this was what we should expect. The reason we had a result lower than expected, in our opinion, is due to the simplicity of our model design.

Taking the measure of using multiple classification models is certainly good in the sense we utilized most of the power we had. However, it is the implementation of each model and the final summation of their results that might be the major flaw in our design. Each

model was tuned by only a few trails and may not be set to their optimal values. Also, the final summarization of the models results was a simple majority vote and we think the results could be better summarized by giving them corresponding weights. In addition, in the case of a tie vote, we chose to always take the classification of the support vector machine (SVM) because of its high reliability in our trails. But this is not always the case, like other classifiers, support vector machine classifier can make mistakes too and because we didn't spend enough time fine-tuning the parameters, this is even more likely.

In conclusion, through this assignment/competition, we had a better understanding of what we have learned in the past semester and how these methods were not merely used for tests and theories but magical real-life applications. More importantly, we learned that hard work and a good attitude is the key to success. If there were another opportunity in the future for such events, we would definitely try harder and exceeds our exception of ourselves.

Reference:

templateLinear function:
http://www.mathworks.com/help/stats/templatelinear.html
templateSVM function:
http://www.mathworks.com/help/stats/templatesvm.html
templateKNN function:
http://www.mathworks.com/help/stats/templateknn.html?refresh=true
kfoldLoss function:
http://www.mathworks.com/help/stats/regressionpartitionedmodel.kfoldloss.html?searchHighlight=kfoldloss
fitcecoc function:
http://www.mathworks.com/help/stats/fitcecoc.html
patternnet function:
http://www.mathworks.com/help/nnet/ref/patternnet.html?searchHighlight=patternnet
Naïve Bayes function:
http://www.mathworks.com/help/stats/templatenaivebayes.html
J.K. Kamarainen, V. Kyrki, H. Kalviainen
Invariance properties of gabor filter-based features-overview and applications
IEEE Transactions on Image Processing, 15 (2006), pp. 1088–1099

Nicol N. Schraudolph, Thore Graepel
Proceedings of the International Conference on Neural Networks, ICANN 2002, pp. 1351–1356