Facial Expression Prediction

CSC411 Machine Learning - Assignment 4

Group: Midnight Mercenaries
Names : Wu Ka Ho (Jeff) and Diego Santos
Student Number: 1000122556 and 1001087138
Kaggle Username: c4santou

Introduction

Hereby we will present our methodology for detecting facial expression with given data set. In this project, we are supplied with 2925 labeled image. Despite we have 98058 unlabeled images, we decided to focus on supervised learning with the labeled dataset. Our objective is to classify test data with the highest accuracy we are capable of.

Approaches

Method 1: Implementation of SVM in Matlab

Our first approach is to build our own multi-class SVM. In the beginning, we separate the data into 7 groups (with the help of given labels) where each represents one class. We notice that the amount of each class is unbalanced i.e. there are more "7", neutral, than any other classes. We decided to balance the training data and use only 200 images from each class (32 x 32 x 1400). We also created a validation set to measure our performance, which has 90 examples of each class. To create "one versus all" SVM, we built 21 different pairwise classifiers (the number of possible combinations is 7 choose 2). This method proved unsuccessful (perhaps due to failure of implementation), and we were only able to accomplish average classification rate of 44% with 8-fold cross validation and 41% on test data.

We then try 2 multi-class SVM library. We ran both using our balanced data, and still produce bad results, only 47% with 8-fold cross validation and 43% on test data. Our failure is largely due to the inability to call the necessary functions.

Method 2: Logistic regression in Python

After some extensive research, we found out that numpy in Python provides a package called "sklearn" that really facilitates model training. We first experiment with logistic regression, which gives us 73% on 8-fold cross-validation, and roughly 71% on Kaggle. It is then reasonable to assume that the dataset we are given is somewhat linearly separable.

Method 3: Linear SVM

With the assumption that the data is largely linearly separable, we then try linear SVM. We performed grid search with different setting of c (range from 0.1 ~ 1.0, and increment by 0.1 each time). We then discovered that by setting c to 0.4 and using 'l2' regularization, we reached our best performance of 76 % yet with 8-fold cross validation, and 72 % on Kaggle. Even though 'l2' regularization is not very robust but is computationally

efficient and stable for a relatively large dataset of 2925 images. At this point, we decided to experiment further with numpy to optimize our solution.

Main Approach

After doing some extensive research on facial expression recognition, we found a paper by Matthew N. Dailey (2001), which suggests that we should preprocess the images using Gabor filter, perform dimensionality reduction on feature vectors, and then input the reduced vectors into our SVM classifier. In particular, we found out that using Gabor filter (Konstantinos G. Derpanis, 2007) bank with 5 frequencies (Pi/2, Pi/4, Pi/8, Pi/16, Pi/32) and 8 orientations starting from 0 and differing by pi/8 could help us with edge detection and feature selection. We proceed by creating a 5 by 8 matrix to store our Gabor filter bank (please see ***gabor_filters.mat***), and apply them to all 32x32 pixels of each image. However, 5 x 8 x 32 x 32 totals to a 40,960 x 1 feature vector for each image, which is way too much for us to input into our SVM classifier. We know we have to perform dimensionality reduction to fasten our training process since SVM does not scale too well with huge feature space -- it has computational complexity of roughly $O(n^3)$.

We decided to give PCA a try. If we try to perform PCA on the feature vector itself, PCA has to compute a covariance matrix having dimension of 40,960 x 40,960 for EACH image, which will takes about roughly 6 GB of RAM. This is therefore not an option. We then try to perform PCA on our original dataset to extract the "eigenfaces", which form a basis set of all images used to construct the covariance matrix before feeding them to Gabor filters. With this process, we were able to achieve best performance of 73% with cross validation and 70 % on Kaggle, which is no better than logistic regression! This is possibly due to the fact PCA skips some important features that would contribute to the recognition accuracy of our model.

After wrestling with PCA for a while and reading multiple papers on the same subject, we are certain that we are doing PCA wrong. We decided to experiment with two other methods: downsizing and downsampling the feature vectors. Before we concatenate all resulting 40 gabor magnitudes, we form a 160 x 256 feature matrix and downsize it proportionally to 86 x 137. We concatenate the columns into one feature vector for each image, and our "gaborized" dataset then has a dimension of 11782 x 2925. Since we have roughly ¼ of the original feature vectors, the training time reduced significantly and takes way less space. To our surprise, inputting the gaborized data into our linear SVM yields a performance of 80 % with 8-fold cross validation and 77.5 % on kaggle.

We then tried the second method of downsampling. We downsample each gaborized image vector, which has a dimension of 1024x1, by taking 1 pixel every 3 pixels. The dimensionality is reduced to 352 x 1, which results in a feature vector of 14080 x 1. We fed the dataset of dimension 14080 x 2925 into our linear SVM, and obtained 79.3% with 8-fold cross validation and 76% on kaggle. We reach the conclusion that a simple downsizing would be the best option available to us.

Optimization is next. So far we have been making the assumption that our data is linearly separable. However, as good computer science students, we decided to experiment further with kernel functions to see if any one of them that better fits our training data. We perform a grid search with sigmoid, polynomial (from degree 2~5), and RBF with default setting for each.

Polynomial and sigmoid kernels performed poorly achieving with their best settings respectively 40% and 73% accuracy in training data. Therefore, we discarded both options. On the other hand, RBF performed slightly better than the linear SVM. This is because RBF Kernel is like a low-band pass filter, a prior that selects out smooth solutions for easier classification.

After reaching the conclusion that RBF, radial basis function, performs the best, we did further grid search to optimize the hyperparameters of RBF. We found out that setting penalty term C set to 100 and gamma to 1.0 e^{-4}, we are able to achieve a good result of ***84.5% with 8-fold cross validation and 81.3 % on Kaggle*****,** which ends up to being our best result. Lastly, we try ensemble SVM with bagging (Hyun-Chul Kim, 2002) using 50 RBF SVM estimators. We obtained very similar results to just one RBF SVM, but it took 5~6 times the time to train.

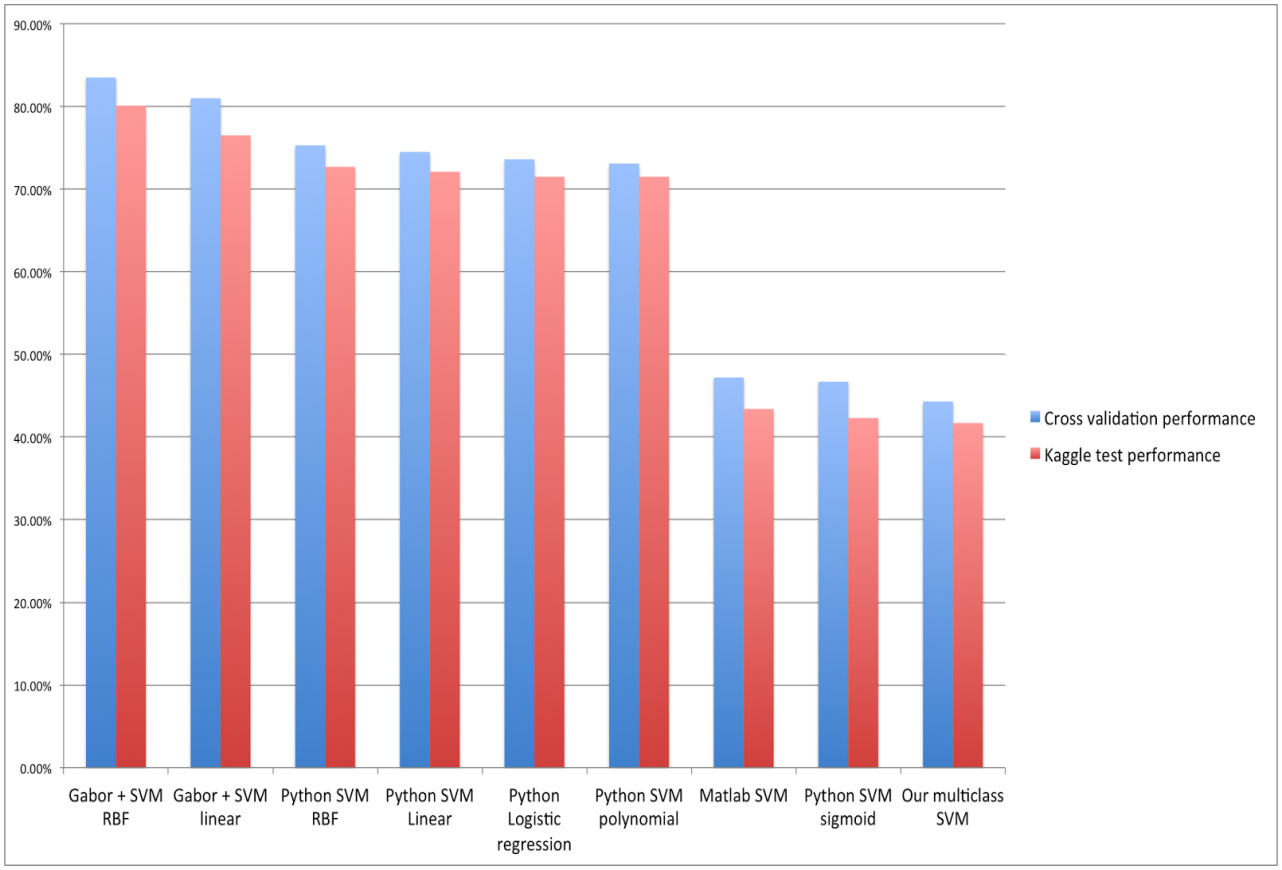
Results

We came to the conclusion that Gabor filters + RBF SVM works the best. In this table below, we show the empirical results of each one of our approaches:

Approaches	Training data (8-fold cross validation)	Test data
Our multi-class SVM	44.3%	41.7%
Matlab SVM library	47.2%	43.4%
Logistic regression	73.6%	71.5%
SVM Linear	74.5%	72.1%
SVM RBF	75.3%	72.7%
SVM sigmoid	46.7%	42.3%
SVM polynomial	73.1%	71.5%
Gabor + SVM linear	81%	76.5%

Gabor + SVM RBF final	84.5%	81.3%
Baseline (kNN)	57%	N/A

This chart provides a summary of our model's cross validation performance and test data/Kaggle performance:



Conclusion

In this project, we focused on the use of SVM along with image preprocessing. When we create the massive Gabor feature vector for each image, if time, memory, efficiency are not a concern, we could have trained on the complete dataset of 40960 x 2925 images to get the best out of gabor filters. The next best thing we could have done is to perform PCA on feature vector correctly. If we can do so, we will be able to choose the principal features that not only speed up our training process, but also potentially give us a better accuracy on the given test dataset and a better generalization on unseen dataset. However, our eventual decision to downsize/downsample our gabor vector isn't a terrible one. It stills gives good, albeit not optimal, results.

Even though we are able to attain reasonable results, there are problems that we could have solved to further optimize our model. One problem is the implementation of multi-class svm in sklearn library. According to its documentation, classification is done in sequential order, and will stop early if the classifier fixate a image to a particular class. Suppose the multi-class svm classify our image to be class 6 when the class should be 7, because of hard assignment, the image would never be classified as class 7 and thus multi-class SVM would gives us an incorrect prediction. The situation would be better if there is soft assignment/probabilistic process where each image is assigned 7 different probabilities, and the decision function simply puts each image into the class with the highest probability.

Another problem we came across during our experimentation is the uneven distribution of facial expressions. For instance, we notice that in both public test data set and labeled image data set, the amount of "7" (neutral) outweighs any other class. Our attempt to "rebalance" involved extracting 200 images from each class and form a dataset of 32 x 32 x 1400. However, this may not be the best idea because we would have less data to train and validate on, which means that we could be skipping important features that are critical to a good model. In addition, from an intuitive standpoint, classifying neutral faces would usually be the hardest because a simple transformation of one particular feature like smiling would completely change the class the image belongs to. In this sense, having the most amount of images belonging to class "7" better prepare our model for classifying neutral faces.

Due to time constraint, we have yet to try other methods that may further improve accuracy. If we did perform PCA on the Gabor feature vectors, we could have use Linear Discriminant Analysis (Yanwei Pang, 2004) to validate how good our feature vectors are. We also thought about using other image preprocessing techniques like Canny edge detector, and other dimensionality reduction like Kernel PCA and multilinear PCA. When it comes to classifiers, we think classifier like Convolutional Neural Network (Steve Lawrence, 1997) could perform better. Other than the fact it would take a long time and significant resources to train a good neural network, and we are yet to be equipped with the knowledge to develop and fine tune one. Another possibility is Adaboost (Ji Zhu, 2009) using weaker classifier coupled with Haar features detection. We figured out how to extract Haar features (Jacob Whitehill, 2006) and use Adaboost, but we don't have the time to develop a full model.

Bibliography

Dailey, Matthew. "PCA = Gabor for Expression Recognition." January 1, 2001. Accessed November 29, 2014. http://cseweb.ucsd.edu/~gary/pubs/pca=gabor.pdf.

Derpanis, Konstantinos. "Gabor Filters." April 23, 2007. Accessed November 29, 2014. http://www.cse.yorku.ca/~kosta/CompVis_Notes/gabor_filters.pdf.

Pang, Yanwei. "A Novel Gabor-LDA Based Face Recognition Method." In *Advances in Multimidia Information Processing - PCM 2004*, 352-358. 1st ed. Vol. 3331. Springer, 2005.

Kim, Hyun-Chul. "Support Vector Machine Ensemble with Bagging." January 1, 2002. Accessed November 29, 2014. http://www.dmli.info/tl_files/paper_upload/SupportVectorMachineEnsemblewithBagging.pdf.
Lawrence, Steve. "Face Recognition: A Convolutional Neural-Network Approach." January 1, 1997. Accessed November 29, 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=554195.
Zhu, Ji. "Multi-class AdaBoost." January 1, 2009. Accessed November 29, 2014. http://dept.stat.lsa.umich.edu/~jizhu/pubs/Zhu-SII09.pdf.
Whitehill, Jacob. "Haar Features for FACS AU Recognition." January 1, 2006. Accessed November 29, 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1613004.

http://www.mathworks.com/matlabcentral/fileexchange/33170-multi-class-support-vector-machine
http://www.mathworks.com/matlabcentral/fileexchange/39352-multi-class-svm

Special Instructions

1) run_model.py: This file takes our model "trainedModel.pkl" and returns a 1 of K matrix of predictions given test data.

!!! IMPORTANT NOTE !!! Before running run_model.py, ***please run the function ImageGaborize.m on the test data to gaborize and preprocess***. The test data will then be saved as "GaborizedImages.mat". The predictions, after running the model, will be stored in "test.csv" and "test.mat".

2) solution.py: this is our final approach. It assumes that the test data is composed of "hidden_test_images" and "public_test_images". It will generate a model named "trainedModel.pkl" and perform 8-fold cross validation on the trained model. The predictions will be saved as "solution.csv" and "solution.mat".

!!!IMPORTANT NOTE!!! Before running this code, ***please run the Matlab file gaborize.m***, which will create gaborized images for "labeled_images.mat", "public_test_images.mat", "hidden_test_images.mat" named "TrainGaborized.mat", "PublicGaborized.mat", "HiddenGaborized.mat" respectively.