ENEE633 Project Report
SVM Implementation for Face Recognition

Ren Mao
School of Electrical and Computer Engineering
University of Maryland
Email: neroam@umd.edu

Abstract
Support vector machine (SVM) is a very popular way to do pattern classification. This paper describes how to implement
an support vector machine for face recognition with linear, polynomial and rbf kernel. It also implements principal component
analysis and Fisher linear discriminant analysis for dimensionaly reduction before the classification. It implements svm classifier
in MATLAB based on libsvm interface as well as scaler and parameter selector, which uses cross validation to find the optimal
parameters for each kernel of classifiers. To apply svm on multiclass problem, it applies the one-against-one method due to its
simplicity and good performance. It measures and compares different classifiers performance in terms of accuracy and gives the
best parameters with the cross validation accuracy for each kernel under different scenarios including different selected training
samples such as expressions and illuminations variations, and the amount of training data.

I. INTRODUCTION
Support vector machines (SVMs) have become a widely studied and applied classification technique, especially used in
face recognition. In order to find out the best practices of SVM classifiers, we need to test different scenarios and measure
accuracy under best parameters. In this project, I investigate various ways to implement SVM classifiers for face recognition
and compare the differences of them.

Specifically, the implementation is based on Matlab and libsvm interface, which provides multiclass classification with
different kernels and cross validation for parameter classification. Firstly I implement a scaler to do preprocessing on train and
test data. Secondly I implement PCA and LDA to reduce the dimension of input data, which is optional. Then I implement
a grid-search algorithm to find the best parameters for different kernels under different scenarios. For each possible candidate
parameter, I test 5-fold cross validation on train data to evaluate the performance of models. With the best parameters, I use
the training data to get an optimal SVM model and test it with the test data to get the final result.

This article mainly i) describes how to scale the training data and testing data for preprocessing, ii) describes how to run cross
validation to select the best parameters for linear, polynomial and rbf kernel SVM, and iii) describes the performed experiments
and their results as well as the comparison of different classifiers. The article is organized as follows: section II describes how
to employ the different methods of classification and the ways to select paramters. Section III provides the experiments result
under different scenarios in table and figure, and compares all these methods. Section IV gives the conclusion of this project.

II. METHODS EMPLOYED
In this project, I employ the SVM classifier on Matlab (version 2013a) with libsvm interface. The procedure is as flowchart
in Fig.1.

Firstly, I split the original data into train set and test set according to selected index for test set. Then I use the train set
data as input to do dimensionality reduction, which includes PCA, LDA or no change. With the weight matrix computed in
PCA/LDA, I transformed the test data set into the same space as train set. On different transformed data(or the original data)
set, I conducted preprocessing to scale the train data and test data into [0, 1] range. With the scaled train data, I run grid-search
algorithm to select the optimal parameters for each SVM with linear/polynomial/ rbf kernels. Then I use the best parameters to
get the svm model and compares the accuracy on test data. Therefore, I could have three classifiers under three transformation
ways which gives nine different methods to employ the face recognition.

A. Preprocessing Scaler
Scaling before applying SVM is very important. The main advantage of scaling is to avoid attributes in greater numeric
ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation.
Because kernel values usually depend on the inner products of feature vectors such as linear and polynomial kernel, large
attribute values might cause numerical problems. Therefore, I linearly scale each attribute to the range [0,1].

\[
\text{mi} = \min(\text{traindata} ,\_1) ; \\
\text{scale} = \text{spdiags}(1./\max(\text{traindata} ,\_1) - \min(\text{traindata} ,\_1) ) ,0, \text{vec\_size} , \text{vec\_size}) ; \\
\text{traindata} = (\text{traindata} - \text{repmat(mi, size( traindata ,1))})\ast \text{scale} ; \\
\text{testdata} = (\text{testdata} - \text{repmat(mi, size( testdata ,1))})\ast \text{scale} ;
\]
Notice that the testdata should be scaled with the same parameters as traindata so that the samples are in the consistent space.

B. Model Selection

In this project, I compared three common kernels in SVM: linear, polynomial, rbf kernels. After decided the kernel, the parameters should be selected.

1) Linear/Polynomial/RBF Kernel: The most trivial kernel is linear kernel, which actually directly uses the inner products of original data. This is useful when the features are already very large. The parameter to be determined is only the cost parameter $C$.

Polynomial kernel maps the data into higher dimension space, which could better describes the classification of data samples. Besides the cost parameter $C$, it also has another parameter $d$, which is the degree of polynomial kernel.

In general, the RBF kernel is a reasonable first choice to train the model. This kernel nonlinearly maps samples into a higher dimensional space so it, unlike linear kernel, can handle the case when the relation between class labels and features are nonlinear. Another reason is the number of hyperparameters which influences the complexity of model selection. The polynomial kernel has more hyperparameters than RBF kernel.

For RBF kernel, we need to determine two parameters $(C, \gamma)$. And if the features are very large, it may not be suitable for using RBF kernel, while linear kernel should be directly used.

2) Cross-validation and Grid-search: For each kernel, we have one or two parameters unknown. In order to find the best parameters under given train data, we have to use some parameter search algorithm. The goal is to identify good parameter so that the classifier could accurately predict unknown test data. The common way to do this is to use cross-validation strategy.

In $v$-fold cross-validation, we first divide the training set into $v$ subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v-1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified.

As the parameters are unknown, I use grid-search method on potential parameters sets using cross-validation. Various pairs of parameters values are tried and the one with the best cross-validation accuracy is picked. And the candidates of parameters are exponentially growing sequences.

The following is piece of code for grid-search:

```plaintext
max = 0;
options = [ option, kernel ];
for c = crange
    for p = prange
        parameters = sprintf('c=%f',c);
        if kernels == 0
            parameters = sprintf('c=%f',c);
        else if kernels == 1
            parameters = sprintf('c=%f,d=%f',c,p);
        else if kernels == 2
            parameters = sprintf('c=%f,d=%f',c,p);
        else
            parameters = sprintf('c=%f,d=%f',c,p);
        end
        f = cvfolds( options, parameters );
        if f > max
            max = f;
            best = parameters;
        end
    end
end
```

Fig. 1. Flowchart of Classification
C. LIBSVM: One-against-one multiclass approach

In the libsvm, the one-against-one method is used to implement multiclass. This method constructs $k(k - 1)/2$ classifiers where each one is trained on data from two classes. For training data from the $i$th and the $j$th classes, it will solve the binary classification problem. After those classifiers are constructed, the 'Max Wins' voting strategy is used to predict the result. In another words, it will vote for the result class in each binary classifier and predict the data $x$ is in the class with the largest vote.

Note that the libsvm implements the one-against-one approach in the way that all $k(k - 1)/2$ classifiers are using the same parameters.

The following code is the typical way of using libsvm, where RBF kernel with parameters $(C, \gamma) = (1, 2^{-5})$ is used:

```matlab
options = '-q -s 0 -t 2 -c 1 -g 0.03125';
model = svmtrain (trainlabel , traindata , options);
[predictions , acc , ] = svmpredict (testlabel , testdata , model , 'q');
```

D. Principal Component Analysis

Principal Component Analysis is always used to reduce the dimensionality of original data set, the basic approach is to compute the eigenvectors of the covariance matrix and then project the original data to the space consist of those eigenvectors. The transformed feature vector is the projection coefficients. Since here sample in all the class are from the same group, which is face of human, it is appropriate to use the whole training dataset to do PCA.

```matlab
u = mean(train,2);
w = train - u*ones(1,num+wn);
 [vec,d] = eig(w'*w);
 [d , sorted] = sort(diag(d) , 'descend');
 vec = vec(:,sorted);
k = 1;
while k < vec_size-1 && sum(d(1:k))/sum(d) < threshold && d(k+1) > 0
 k = k+1;
end
vec(:,k+1:end) = [];
```

As the above code describes, I sort the eigenvectors as descending of eigenvalues and choose the top vectors, which the energy reaches the threshold(0.95), as basis to transform all data into lower dimensional.

E. Fisher Linear Discriminant Analysis

Other than the PCA, LDA produces an optimal linear discriminant functions which maps the data into classification space by taking the class distribution into consideration. The basic approach is to find the generalized eigenvalues and eigenvectors as following form:

$$S_B w_i = \lambda_i S_W w_i$$

(1)

where, $S_W$ is the with-in class scatter matrix and $S_B$ is the between-class scatter matrix. Once again, the issue due to small training sample set is that the $S_W$ is singular which may gives incorrect eigenvectors if we just use `eig` function in Matlab to compute the generalized eigenvalues.

The way I do here is to use a two-phase projection: 1.) find the orthogonal basis of $S_W$ and project the data into this non-null space respect to $S_W$; 2.) in that space, recalculate the scatter matrices and do normal LDA, where the $S_B$ will be full-rank and lead us easy to get eigenvectors.

```matlab
if (rank(sw) < vec_size)
fprintf (' Projecting data into non-null space due to singular Sw\dots\n');
T = orth(sw);
train = cellfun (@(x) T.'*x, train , 'UniformOutput',false);
 test = cellfun (@(x) T.'*x, test , 'UniformOutput',false);
```
III. Experiments and Results

A. Classifier Comparison

To compare the SVM with different kernels I implemented, I choose the first half (lower integer) of all data set as the testing data and others are training data to get the accuracy results as Table I and Fig. 2:

<table>
<thead>
<tr>
<th>Method</th>
<th>data.mat</th>
<th>pose.mat</th>
<th>illumination.mat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ori</td>
<td>PCA</td>
<td>LDA</td>
</tr>
<tr>
<td>Linear</td>
<td>65%</td>
<td>77%</td>
<td>80%</td>
</tr>
<tr>
<td>Polynomial</td>
<td>61%</td>
<td>69%</td>
<td>81%</td>
</tr>
<tr>
<td>RBF</td>
<td>61%</td>
<td>68.5%</td>
<td>80%</td>
</tr>
</tbody>
</table>

From the results, we could see that:
- When the number of features are much larger than the number of samples, which would happen in the original data set, the performance of linear kernel would be better than polynomial and RBF kernel. This is because of features are enough and no need to map into higher dimension.
- When we mapped the data into lower dimension by PCA/LDA, the rbf kernel and polynomial kernel are usually better than linear kernel since the features are much less and might be necessary to be mapped into nonlinear space.
- The performance of classifiers in LDA data set are usually better than original/PCA dataset since LDA would transform data into space helps to classification. Note from the results, the pose data set would give us an opposite performance, where the SVM on original dataset has better accuracy. This might be resulted from the side effects of dimensionality reduction.

B. Effects of Training Data

To investigate the effects of training data, I firstly iterate to see how the classification accuracy varies as the amount of training data is changed. Fig. 3 shows the accuracy of classification in pose dataset. From this figure, we can see that the
- The accuracy is apparently increased as the amount of training data is increased for all different kernel SVM classifiers.
- As the samples are increased, the SVM on original data set is better than PCA/LDA dataset. This might because of information loss and noise addition while doing PCA and LDA.
It can be seen that the SVM on LDA data set is generally better than PCA dataset since LDA transforms data into space which helps classification.

Then, I also investigate the effects of selected training data by changing the partition of data.mat set. In the data.mat set, we have neutral face, expression face and illumination face. So I fixed the amount of training samples as two, choose each of the image as test sample, others as training samples to see how the accuracy varies. The Table II shows the results of different test sample.

<table>
<thead>
<tr>
<th>Test Sample</th>
<th>Linear</th>
<th>Polynomial</th>
<th>RBF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ori</td>
<td>PCA</td>
<td>LDA</td>
</tr>
<tr>
<td>Neutral Face</td>
<td>61%</td>
<td>77%</td>
<td>80%</td>
</tr>
<tr>
<td>Face with expression</td>
<td>58%</td>
<td>64.5%</td>
<td>69%</td>
</tr>
<tr>
<td>Face with illumination</td>
<td>60.5%</td>
<td>69.5%</td>
<td>49.5%</td>
</tr>
</tbody>
</table>

From the table, we can figure out that:

- For SVM classifiers, it is better to take the variation samples as training samples since their common features may be still helpful for classification. And thus the accuracy would be better then using neutral face as training samples.
- For linear kernel SVM, the performance after LDA is better than original dataset except we take face with illumination as test sample. This might because of the characteristics of training sample would be sensitive to illuminations for support vectors.
- For LDA dataset, the rbf kernel SVM performs almost exactly the same as linear kernel SVM. This might because that the LDA makes the data almost linear separable where linear kernel and rbf kernel are equivalent.
- When we take the illumination sample as one of training sample, the LDA accuracy may be much better than PCA accuracy, which makes sense since the PCA is to project data according to variation direction and is not sensitive to the global change such as illuminations.

IV. CONCLUSION

In this project, I implement and compare different methods of SVM classifiers: linear, polynomial, rbf kernel and also PCA and LDA. To correctly apply SVM, I implement preprocessing scaler to avoid the numerical dominance issues. Also I implement grid-search algorithm to select parameters for each SVM based on 5-fold cross-validation. I performed several experiments on different datasets to compare the classifiers and test the effects of variations as well as amount of training data. The result shows that different classifier may have different properties. Usually, the rbf kernel SVM is better then others on low-dimensional feature space such as data after PCA/LDA. Nevertheless, the linear kernel is good enough for the original dataset since the number of features are large enough and the parameters to be determined are less then rbf/polynomial, which could save computation time.

REFERENCES


