C S 391L: Machine Learning Assignment 4 **Julian Sia (EID: JAS7794)**
Prof. Dana Ballard Due Date: 08-Apr-16

Classifying Handwritten Digits with Support Vector Machines (SVMs)

This report provides a brief description of the SVMs and experimental observations of the SVM algorithm in classifying hand-written digits.

SVM Algorithm: Training Methodology

Reshaping

In the training phase of the Eigendigits Algorithm, we first take an n_{train}-sized training set of x by y 1-channel images. We then flatten each x by y image into a k dimensional vector to produce matrix A, a k by n matrix of training examples.

Preprocessing: Normalization and PCA

Using this 784 by n_{train} matrix we calculate the transpose of this matrix, forming a n_{train} by 784 matrix, a format that is acceptable by the implementation of SVMS we use in `sklearn`. After this matrix transposition, we normalize the pixels of each image (each row in the matrix corresponding to a 784 dimension training example) using the standard Euclidean norm. By construction of the SVM, which uses a max margin hyper-plane to separate data, normalizing allows the SVM to classify digits properly. Failure to normalize the data results in an extreme degradation in performance. We also experiment to see if applying PCA as a dimensionality reduction technique helps the SVM.

Fitting the SVM

One vs. Rest or One vs. One

After preprocessing the data, we proceed to use the one-vs-one approach to the multi-class classification problem, which trains $\frac{K(K-1)}{2} = \frac{10(9)}{2} = 45$ binary classifiers for $K = 10$ classes. This approaches uses a setup in which a majority vote of the classifiers determines the label. Although this approach may seem more costly than the one-vs-rest approach in which a classifier is trained for each digit class (0 through 9) and the relative real-valued margin of confidence between each classifier is compared, with the most confident classifier's label being chosen, the literature has claimed that it may increase training time given that each classifier only need to be trained on a smaller subset of data. We evaluate the training time of the algorithms to see if this advantage exists.

Testing Methodology

Reshaping

Taking a n_{test} sized testing set of x by y 1-channel images, we create matrix B of dimensions n_{test} by 784.

Preprocessing

We then apply the same preprocessing as in training, using the Euclidean norm to normalize each 784 dimensional testing example.

Classification

After preprocessing, we proceed to use our one-vs-one or the one-vs-all multi-class SVM to determine the appropriate class label for each testing example. In some sets of the experiments, we apply PCA after preprocessing to produce a reduced dimensional representation to see how training on this reduced-dimensional representation affects classification performance.

Kernel Functions

When solving the computationally easier dual problem for SVMs, there involves some sort of comparison of a test vector x_j relative to a set of training vectors x_i, where $i = 1$ to n. This is accomplished with a dot product, but in the case that data may not be linear separable in the given vector space it occupies, it is useful to project the training vectors and test vector into another space where they may be separable, applying some $\Phi(.)$ such that $\Phi(x_i)$ and $\Phi(x_j)$ are computed and the dot product $\Phi(x_i) \cdot \Phi(x_j)$ is then computed. This in itself a costly operation. Kernel functions $k(x_i, x_j)$ that accept each x_i and the x_j provide an equivalent, less costly transformation such that $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. We test the linear and radial basis function kernels to see how well they aid in the separation of data.

Evaluation Metric

We use a simple classification accuracy metric by dividing the number of correctly classified testing images over the total number of testing images.

Implementation

I implemented my SVM experiments in Python 2.7 using NumPy, SciPy, and Scikit Learn packages. I wrote a variety of functions to run experiments testing the SVM. The Function `PCA_SVM` is responsible for running PCA on a Euclidean normalized training set to do dimensionality reduction, training on the reduced training representation, and testing on an also dimensionality-reduced testing representation. The function `tune_SVM` is used to evaluate the performance of *linear* and *rbf* kernels in aiding the SVM in the digits classification task, accounting for tuned parameters. The function `SVM_vary_train` provides the same functionality as `tune_SVM` but accommodates for different training data size.

Experiments and Discussion

Baseline

We define our training baseline as the $F1$ score of the one vs. one classifier trained on the full 60000 training set of 784 dimensional training examples normalized with the Euclidean norm, using a linear kernel.

Training Data and Kernel Discussion

Training Size Experiments

As in previous experiments with PCA-KNN, we note that the inclusion of more training data (seen in the table in the left below) used to train the SVM results in a better testing performance as measured by F1. These results are sensible given that the SVM has more training data to construct max margin hyperplane.

Training Size	**F1**
60000 (linear kernel)	.9460
10000 (RBF kernel)	0.9603
20000 (RBF kernel)	0.9687
30000 (RBF kernel)	0.9741
40000 (RBF kernel)	0.9764
50000 (RBF kernel)	0.9782
60000 (RBF kernel)	0.9794

Gamma	**F1**
RBF ($\gamma = 10^{-3}$)	0.1135
RBF ($\gamma = 10^{-2}$)	0.8477
RBF ($\gamma = 10^{-1}$)	0.9451
RBF ($\gamma = 10^{0}$)	0.9794
RBF ($\gamma = 10^{1}$)	0.9487
RBF ($\gamma = 10^{2}$)	0.1135
RBF ($\gamma = 10^{3}$)	0.1135

Penalty	**F1**
$C = 10^{-4}$	0.1135
$C = 10^{-3}$	0.7108
$C = 10^{-1}$	0.9348
$C = 10^{-0}$	0.9460
$C = 10^{-1}$	0.9477
$C = 10^{-2}$	0.9401
$C = 10^{-3}$	0.9308

Kernel Experiments

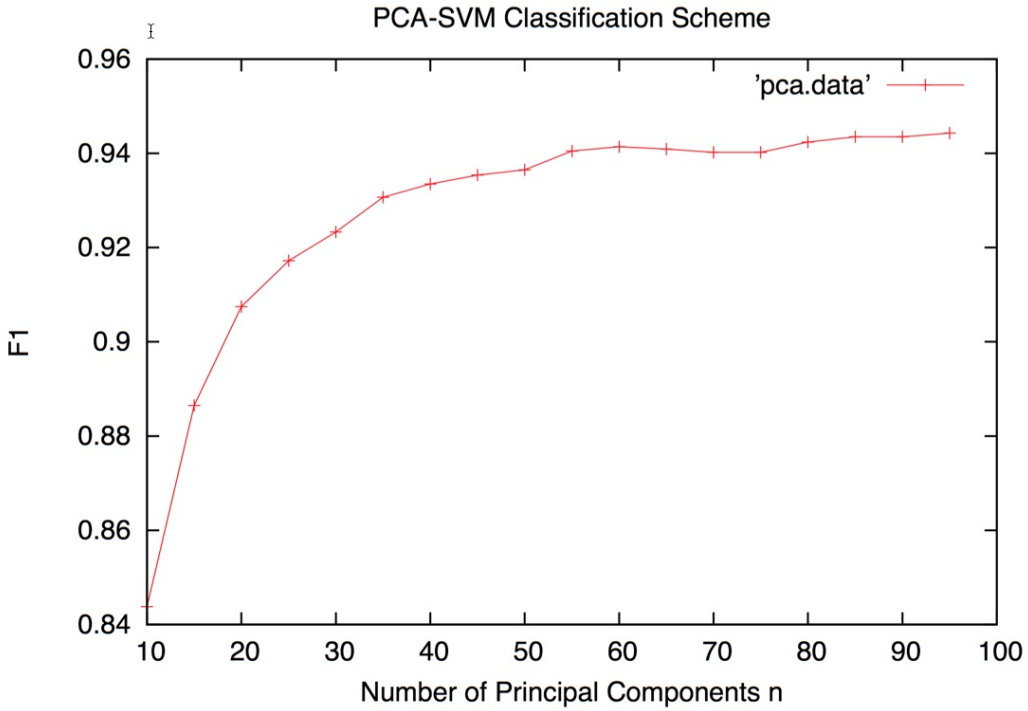
In the Kernel experiments, as indicated in the second table, we also make the observation that the RBF kernel outperforms the linear kernel at testing time. We note that higher performance of the RBF kernel may be attributed to the fact that simple linear separators of the data are sub-optimal relative to non-linear decision boundaries induced by the rbf kernel. However, it is important to take into account that given the high dimensionality of the data, a linear separator will be sufficient. Given the curse of dimensionality, the relative distance between points in a 784 dimensional space becomes large, and thus a linear separator is efficient in separating the data. This may be a plausible reason why the linear-kernelized SVM performs only slight worse than the rbf-kernelized SVM. In addition, the computational cost versus F1 performance tradeoff may weigh in favor of using a linear kernel. With additional parameter tuning on C, the penalty for SVM misclassification, the multiclass SVM using a linear kernel could most likely perform better than the RBF kernel. Attempts to train a polynomial-kernelized SVM were made; however, due to technical difficulties with training, the results were not included.

Penalty (C) Experiments

In this experiment, we adjusted the C parameter, which corresponds to the misclassification penalty for the SVM. For low values of C we notice that performance severely degrades, which corresponds to the SVM looking for an extremely large margin separating hyperplane, at the expense of misclassifying points. For high values of C, we notice that performance takes a slight downturn, which corresponds to the SVM making a smaller margin separating hyperplane not penalizing incorrectly classified examples during SVM training.

PCA-SVM Experiments

In the spirit of the last assignment, we wished to see if doing further dimensionality reduction on the data would allow for a more compact representation of digits that would yield comparable performance. Following the direction of the PCA-nearest-neighbors approach yielded in the last assignment, we examined the performance of a linear kernel SVM using our default one-vs-one multi-class scheme.



n	F1
n=10	.8438
n=15	.8865
n=20	.9075
n=25	.9172
n=30	.9233

n	F1
n=35	.9307
n=40	.9335
n=45	.9354
n=50	.9365

n	F1
n=55	.9405
n=60	.9414
n=65	.9409
n=70	.9402

n	F1
n=75	.9402
n=80	.9424
n=85	.9435
n=90	.9435
n=95	.9443

To a certain degree, performance degrades after applying PCA on the digits data. This is an artifact of PCA, which essentially results in information loss (as we are discarding the eigenvector components which capture the covariance in the digits corresponding to lower valued eigenvalues). Though these top n components probably capture the greater degree of the variance in the digits data set, throwing away the rest of the components aside from the chosen n still results in information loss, hence the slight performance drop.

From the graph, we can clearly see that approximately 35 eigenvectors capture a large degree of the variance. Though there may be performance gaps, PCA may provide an efficient (albeit lossy) encoding of the digits data, which is borne out in the SVM classification.

Performance of One Vs. One (ovo) relative to One Vs. Rest Classification (ovr)

To empirically evaluate the performance of one-vs-one (ovo) and the one vs. rest (ovr) multiclass svm, we evaluate the runtime to train both a ovo and ovr classifier. Interestingly, the performance was nearly equal for both choices of the multi-class svm.

SVM Type	Train Time
OVO	408.92 user, 3.04 system 6:49.65 elapsed 100%CPU
OVR	410.11 user, 3.01 system, 6:50.70e lapsed 100%CPU

Future Work

Finding the Optimal Parameters

It is important to note that we make no claims regarding the optimality of the above hyperparameters. In order to do proper hyperparameter tuning it is important to split the training set, in which one portion is used to train the SVM and the other set (the validation set) is used to tune the hyperparameters. After the hyperparameters are tuned, only then should testing be undertaken. A useful future task could be using this methodology to determine the optimal hyperparameters.

Attachments

`sia_hw4_code.zip`

References

[1] Bishop, Christopher J. 2006. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006 [cited 15 February 2016].