Machine Learning 2014, Hand In 2

Version 2

Intro

In this exercise you will build more classifiers for optical character recognition (OCR). The main exercise is building a classifier for the AuDigits that you yourself has created. The goal is simply to do as well as possible on the 1000 unknown images i have kept secret. When you hand in be sure to write how well you think your classifiers will work on the 1000 unknown images e.g. tell you customer how good you expect your algorithm is. For all experiments you must include the results you get on the MNIST digits and discuss any large discrepancies. This is a very open/broad assignment where you yourself have to figure out where to put the focus.

Report

You can write your report in Danish if you prefer. The maximal report length is 5 pages. You are allowed to be up to 3 members in a group. You are encouraged to discuss the exercise between groups and help each other as much as possible without of course copying each others work. Particularly, discussing the quality of your classifiers is probably a good idea to get an indication if you are doing it correctly. Be sure discuss all the choices you have made in your implementation in your report and explain why your think the quality of your classifier is as it is. Bonus exercises may be skipped. When you hand in your report remember to include the matlab files we asked for in the exercises and all the supporting code to use them.

There are associated data files and matlab examples for you on the website. The files are....

mnistTrain.mat a matlab file that contains the MNIST digits

mnistTest.mat a test set for MNIST digits

auTrain.mat a train set with images you produced comes online when we have parsed your inputs

minFunc.zip A library that includes very efficient optimization algorithms that you can use

softmaxTrain.m,softmaxPredict.m The softmax model from hand in 1 that uses minFunc to reduce the running time

AutoEncoder.zip Files for the sparse Autoencoder discussed in class, including autoEncoderCost that implements the backpropagation algorithm for the sparse autoencoder. Includes running example you can look at and run.

Ideas to deal with small data set.

You should use regularization and validation for model selection (e.g. selecting any form of hyper parameters) for all the classifiers you train. But there is still the issue that the data set for AuDigits is very small compared to the dimensionality of the images. There are two approaches for you to try.

Generating More Data

You can generate more data by considering the task we are given. A picture of a 5 is still a 5 even if add a little random noise to the image, or if you stretch, scale or rotate the image slightly. This gives you many possible ways of generating new data. Matlab has built in commands for manipulating images you should take advantage of. For instance *imnoise* adds noise in they you specify to images, *imrotate* rotates an image and *imtransform* does general transformations (scale,translate,stretch,...). Use help or doc command to learn about their functionality. See `http://yann.lecun.com/exdb/lenet/index.html` for more inspiration if needed.

Dimensionality Reduction/Pretraining

As discussed in the lecture we can reduce the input dimensionality while still having most of the information intact. Matlab has an implementation for PCA (princomp in matlab 2012) you can use for this purpose directly. You should also try the sparse autoencoder (explained later) from the notes that i have provided the code for. You should try different settings for the parameters that can be used. You can try to use PCA and/or the sparse autoencoder on the MNIST digits to find a good set of features that you can map the AuDigits data to see if that helps. Experiment and experiment.

Support Vector Machines for OCR

In this exercise you will experiment with support vector machines and see if you can improve on the linear classifier from hand in 1. You should download and install the free SVM toolbox libsvm `http://www.csie.ntu.edu.tw/~cjlin/libsvm/` (the matlab version) and use. You should use the SVM type C-SVM. This is the SVM version where points are allowed to be on the wrong side of the margin but the distance to the margin is penalized by a constant factor C that must be set. There are two important functions, svmtrain and svmpredict. svmtrain returns a model object that is needed for svmpredict.

SVM Code Write code such that you can apply a SVM to your input data and get a model back and use it to do prediction on new images. The model object contains a list of the support vectors. Use validation whenever you have to choose between several parameters (libsvm has cross validation built in).

Linear Kernel Try the linear Kernel with different values for the cost of being on the wrong side of the margin, C.

Polynomial Kernel Try the polynomial Kernel with different polynomial degree d and C

RBF (Gaussian) Kernel Try RBF Kernels with different shape parameter, γ, and C.

Deliverables In the report, shortly explain what you have done and the results you get for all the different Kernels, both in sample and out of sample (for MNIST), and what behavior you expect your classifier to have out of sample on the hidden data for AuDigits. I expect you to generate more data for the AuDigits and/or use the dimensionality reduction techniques to try and boost your classifier for the AuDigits. When you train your algorithms for the MNIST digits the test data is not a validation set but a test set. If you use the test set to pick the best classifier between different parameter settings you polluting your out of sample estimate.

Save the best (over all kernels, parameters etc) SVM for MNIST and AuDigits you have found in a bestSvm.mat file with the save command. If data (particular AuDigits) needs any preprocessing (like dimensionality reduction or rescalings) before being used by the saved SVM model, save that in file (and function) preprocessSvm.m. Finally, make a function runBestSVM.m that takes as input n images and returns a vector of predictions (this should be optimized for the au digits).

Neural Nets and Autoencoders

I have provided you with an implementation of the sparse autoencoder explained in class. You can use that for dimensionality reductions and combine it with the softmax code or SVM and to learn good classifiers.

Deliverables Use the autoencoder with the SVM and/or the softmax where you experiment with the size of your dimensionality reduction and what other parameters that comes into play. Explain your results in the report.

Bonus Exercise: Implement a neural net for digit classification

Take a look at the code in the autoEncoderCost.m function that implements backpropagation for the autoencoder. You should be able to alter that to just be a neural net (only the output layer and the sparsity cost should change) for digit classification. You should change the output to 10 neurons and use softmax in the output layer and change the cost function from squared difference to the cross entropy function defined in hand in 1. Look in the softmax file provided for inspiration. I have provided numerical differentiation code that can check whether your cost and gradient implementation is correct (e.g. whether you compute the gradient of the cost).

Neural Net Cost and Gradient Implement backprop in a file named neuralnetCost.m It must return cost and gradient as the autoEncoderCost in a vector.

Verify Correctness Verify that it works with checkneuralcost.m

Use the neural net Train a classifier for the OCR problem for different weight decay parameters and size of the hidden layer (validation) using your neuralnetCost and minFunc. You may of course try to combine this with dimensionality reduction techniques you have used or use any extra data you have generated. Explain what you have done and what you got in the report and compare it with your other results. Do we have a clear winner. Remember to estimate the out of sample error for the AuDigts (this is what we are trying to minimize).

Deep Net (Extra Bonus) If you have implemented backpropagation (neuralnetCost) you can extend it to several layers. Use the autoencoder to pretrain 2 or more hidden layers and use

backprop with softmax to do the final fitting. Here you should only use weight decay on the last level (from last pretrained layer to softmax output). We do not want to destroy our already constructed features only finetune them. Experiment and see what you get.

For this exercise you should save the best nets for MNIST and AuDigits you have found in a mat file bestNeural.mat and any preprocessing of the data needed you put in file preprocessNeural.m. Finally, make a function runBestNet.m that takes as input n images and returns a vector of predictions (this should be optimized for the au digits).