ADVANCED MACHINE LEARNING & PERCEPTION
COMS 4772 & 6772 HOMEWORK # 2
PROF. TONY JEBARA

Due: November 2, 2010 by 4pm
Total Points: 25

Multitask and Meta-Learning

Read the paper *Multitask Sparsity via Maximum Entropy Disrimination* by T. Jebara. We will be using the code available for that paper for Homework 2 and to explore various multitask problems. The code is available here:

`http://www.cs.columbia.edu/~jebara/code/multisparse`

Download a copy to use as a starting point in your work. Note, the defaul quadratic programming solver in Matlab is very slow. It is better to use a much faster solver such as Mosek (or cutting plane methods but these are much trickier to implement for this homework). Also, due to slow quadratic programming routines, it is best to avoid giving it problems with too many variables whenever you call the quadratic program. Follow these steps to get a Mosek installation up and running. Mosek installation is not very hard. Here are steps to follow (on unix/linux). If you follow these steps, quadprog will automatically call the mosek version. If there are any doubts, please visit mosek.com for more details.

```
1. Go to the following page and download
the correct version for your platform
http://www.mosek.com/index.php?id=7
2. Download a free license from the following page
http://www.mosek.com/index.php?id=99
3. Unzip the mosek package you just downloaded.
4. Set the following two environment variables:
MOSEKLM_LICENSE_FILE to the license file you downloaded
5. Set LD_LIBRARY_PATH to
<mosek>/5/tools/platform/linux32x86/bin/
(note that this path may change slightly depending on your platform)
6. Add the following to MATLAB path
<mosek>/5/toolbox/r2007a
```

You will be asked to explore datasets of your own choosing in this homework. If you do not know where to begin, consider the following repositories:

```
http://archive.uci.edu/ml
http://www.ee.duke/edu/~lcarin/LandminData.zip
```

Try to find a good dataset that showcases how multitask learning can help improve

over standard SVMs (i.e. the tasks should be related, otherwise, multitask learning is not beneficial).

It turns out that there are not many multitask datasets out there. Instead, one trick is to use multi-class datasets as a surrogate. Multi-class is not quite multitask. Instead, we only have one task and each input has a label m where $m = 1, \ldots, M$ if there are M classes. To proceed, we can convert any multi-class classification problem into M binary ± 1 classification problems where each binary task is to predict if the input is in class m (then $y = 1$) or otherwise (then $y = -1$). Each binary classification can now be viewed as a different task. Unlike a true multitask problem, however (say adult/child and male/female), only one of the tasks can be positive and all others must be negative. In a true multi-task problem, any binary configuration is potentially allowable. If you can find a true multitask multi-label problem, then your results will be even more interesting. Subsequently, you will explore learning about these binary tasks in isolation (as independent binary classification problems) or in multitask settings where a shared feature, kernel or common classifier couples them.

Note: if your dataset has missing values in some of the attributes, try just putting the mean for the missing value as a quick hack to get started. Also, in many settings, it helps to scale the dataset so that the input features for your dataset (i.e. each dimension) range between 0 and 1. Split your dataset into training and testing components and try to control the training set size so that the algorithms are not too slow.

1 Multitask Feature Selection (8 points)

In this first problem, you will compare the multitask algorithm against independent support vector machines without any feature selection. The multitask method will use feature selection followed by linear classification (so nonlinear kernels are not needed here). Choose any dataset that is either a multi-class classification problem or a true multi-task problem.

We will actually use kernel selection code to perform feature selection. If you have features in D dimensions, i.e. $\mathbf{x} \in \mathbb{R}^D$, form $d = 1, \ldots, D$ kernels as $k_d(\mathbf{x}, \bar{\mathbf{x}}) = \mathbf{x}(d)\bar{\mathbf{x}}(d)$.

For a few values of $\alpha = \frac{1-\rho}{\rho}$ starting with $\alpha = 0$ (which indicates no feature pruning), sweep the value of C. Train and test the multitask algorithm at each value of C and report the total error across all tasks for both train and test data. Plot the total number of training and testing errors as you vary C for several choices of α. Recall that $\alpha = 0$ is just plain independent SVMs. Meanwhile, at different values of α, you will start getting multi-task sparsity and some features dropping out.

Discuss the dataset (its size, features, tasks, and classes), your findings, the plots you generated, and any conclusions you can draw. Show the set of chosen features for a good setting of α and C as a bar plot over the D dimensions of your dataset. Discuss the training and test sizes, running time and any measures you took to get the method to work.

2 Multitask Kernel Selection (8 points)

Perform the same analysis as above but use polynomial kernels. Instead of feature selection, consider $d = 1, \ldots, 8$ kernels which are defined as

$$k_d(\mathbf{x}, \bar{\mathbf{x}}) = (\mathbf{x}^\top \bar{\mathbf{x}} + 1)^d.$$

Once again, discuss the dataset (its size, features, tasks, and classes), your findings, the plots you generated, and any conclusions you can draw. Show the set of chosen kernels for a good setting (i.e. one with low total error) of α and C as a bar plot of $\hat{\mathbf{s}}(d)$ values over the D possible kernels of your dataset. Discuss the training and test sizes, running time and any measures you took to get the method to work.

3 Adaptive Pooling (9 points)

For this problem, you will have to modify the code so that it performs adaptive pooling (instead of feature and kernel selection). Here, assume you are given several datasets (say of face images) that have been labeled in the same way (say as male or female). Note, do not use a multi-class dataset for this because it does not make sense here. The datasets have to be of inputs which are labeled in the same way. In this multi-dataset setting, you could learn an SVM for each dataset or you could do *pooling*. In other words, just learn a single SVM by combining all datasets into one large dataset.

A smarter approach is *adaptive pooling*. There, the datasets $m = 1, \ldots, M$ have to *choose* between using their own specialized classifier θ_m or a communal classifier θ. For each dataset m, define a binary feature selection variable vector $\mathbf{s}(m) \in \mathbb{B}$ which determines if the dataset will utilize its own θ_m or use the θ communal model for making its predictions. Consider $\mathbf{s} = [\mathbf{s}(1), \ldots, \mathbf{s}(M)]$, a vector containing the M binary variables that indicate (with $\mathbf{s}(m) = 1$) if the m'th dataset should use a specialized classifier. We want this vector to be sparse so only some datasets get to specialize and some will have to use the communal classifier. The parameter ρ or α on the Bernoulli prior on datasets can be used to indicate how many of them will be communal and how many will be specialized. Write the maximum entropy discrimination (MED) code to learn an adaptive pooling set of SVMs which figures out if the m'th classifier should use its own model or the communal model. The prediction rule for an input $\mathbf{x}$ for the m'th dataset is given by:

$$\hat{y} \quad = \quad \text{sign}\left(\hat{\mathbf{s}}(m)\sum_{t=1}^{T_m}\lambda_{m,t}y_{m,t}k_m(\mathbf{x},\mathbf{x}_{m,t}) + \sum_{n=1}^{M}\sum_{t=1}^{T_n}\lambda_{n,t}y_{n,t}k(\mathbf{x},\mathbf{x}_{n,t}) + \hat{b}_m\right).$$

where $\hat{\mathbf{s}}(m)$ is in the paper and will be large if dataset m is specialized and will be close to zero if dataset m only needs the communal classifier to work. Implement the resulting MED optimization problem as a sequential quadratic program by using the bound from the paper. Apply this to learn a multi-dataset problem (for example labeling faces as male/female). Here the problem consists of slightly different datasets (over the same input space) which are labeled in the same way. Show the train and test performance of training M separate SVMs with *no pooling*

across values of C for these datasets. Then show train and test performance when *pooling* the data and learning a single SVM across values of C for these datasets. Then show how you can do better (lower total test error) by implementing the *adaptive pooling* method described above (over different values of C and some choices of α). Note, you may want to avoid using kernels to keep things simple (i.e. linear SVMs).

Discuss the datasets (size, features, etc.), your findings, plots you generated, and any conclusions you can make. Show which datasets used their own model or the communal model by reporting the values for the switches $\hat{\mathbf{s}}(m)$ for a low total test error setting of α and C. Discuss the training and test sizes, running time and any measures you took to get the method to work.