Final Report: Semi-Supervised Learning of Pen-Based OCR.

Joshua Brakensiek, Jacob Imola, Sidhanth Mohanty

Abstract

This report explores using semi-supervised learning to classify vector-based representations of handwritten digits. We describe related semi-supervised approaches such as transductive support vector machines (TSVMs). FINAL STUFF

1 Background

Handwriting recognition is a classic machine learning problem. Generally, an image of a digit is inputted to an algorithm and classified. Instead of taking this approach, we used the sets, "Pen-Based Recognition of Handwritten Digits Data Set" [AA98] from the UCI Machine Learning Repository [Lic13], which consist of pixel coordinates that the writers' pens took at certain time intervals. This data set, which has over 10,000 data points, consists entirely of drawings of the digits $0, \ldots, 9$. We seek to explore if adding the extra information of time while sacrificing some detail from the shape of the digits can produce a viable semi-supervised model. We implement our learning algorithms in Python 2.7, using on the NumPy, SciPy, and Scikit-learn libraries (and possibly other libraries as we see fit). The models we try are Transductive Support Vector Machines and graph-based kernel methods and regularization. Throughout this paper, let $X = \{x_1, x_2, \ldots, x_k\}$ be the labeled training data, $Y = \{y_1, y_2, \ldots, y_k\}$ be the corresponding labels, $X^* = \{x_1^*, x_2^*, \ldots, x_n^*$ be the unlabeled training data, and $Y^* = \{y_1^*, y_2^*, \ldots, y_n^*$ be variables representing the unlabeled training data's labels. Now, we will give a description of the models:

Recall that an SVM reduces to the following primal problem:

$$\arg\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{k} \zeta_i \tag{1}$$

$$\begin{aligned} \text{subject to} \quad & \forall_{i=1}^{k} : y_i(wx_i + b) \geq 1 - \zeta_i \\ & \forall_{i=1}^{k} : \zeta_i > 0 \end{aligned}$$

A TSVM's primal problem allows the unlabeled data to take on any label, and hence reduces to a similar problem [Joa99a]:

$$\arg\min_{w,b,\zeta,\zeta^*,Y^*} \frac{1}{2} w^T w + C \sum_{i=1}^{k} \zeta_i + C^* \sum_{i=1}^{n} \zeta_i^* \tag{2}$$

$$\begin{aligned} \text{subject to} \quad & \forall_{i=1}^{k} : y_i(wx_i + b) \geq 1 - \zeta_i \\ & \forall_{i=1}^{n} : y_i^*(wx_i^* + b) \geq 1 - \zeta_i^* \\ & \forall_{i=1}^{k} : \zeta_i > 0 \\ & \forall_{i=1}^{n} : \zeta_i^* > 0 \end{aligned}$$

Here, C and C^* are parameters that control the type of fit we get.

2 Related Work

The TSVM algorithm that we implemented requires that in the unlabeled data, the number of data points classified is constrained to be exactly equal to some parameter num_+. [J$^+$03] shows that this special kind of TSVM is equivalent to solving a s-t min cut problem, where the size of the partitions are fixed beforehand, and explores an algorithm called spectral graph transducer, that does not require the sizes of the partitions to be fixed beforehand.

Closely related to the graph interpretation of semi-supervised SVMs are methods intersecting at spectral graph theory and semi-supervised learning like Laplacian SVMs and Manifold regularization (see [BNS05]). This method assume a distribution P over $X \times \mathbb{R}$ from which labeled examples (x, y) are drawn, and assumes that the marginal distribution of P on X, denoted P_X, has the structure of a Riemannian manifold and uses this model as a basis for a family of methods.

Another related semi-supervised learning technique that could be applied to TSVMs is co-training, introduced in [BM98]. This technique splits each unlabeled example into multiple 'views' (an example as specified in [BM98] would be how webpages can be partitioned into the content on that page and content on other pages that link to it), and two learning algorithms are trained on different views. Then, the labels provided by the algorithms on previously unlabeled examples are used to increase the labeled data available to the other algorithm to train on.

[Joa99b] describes an algorithm called SVM^{light}, which is faster than traditional algorithms for training semi-supervised SVM that scales well with a large amount of data, but sacrifices on accuracy.

3 Methods

3.1 TSVM

One TSVM method we use is an SVM with local search which was first described in [Joa99a]. We use a more generalized version of an SVM:

$$\arg \min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{k} \zeta_i + C_+^* \sum_{i=1}^{n} \zeta_i^* [y_i^* == 1] + C_-^* \sum_{i=1}^{n} \zeta_i^* [y_i^* == -1] \tag{3}$$

$$\begin{aligned} \text{subject to} \quad & \forall_{i=1}^{k} : y_i(wx_i + b) \geq 1 - \zeta_i \\ & \forall_{i=1}^{n} : y_i^*(wx_i^* + b) \geq 1 - \zeta_i^* \\ & \forall_{i=1}^{k} : \zeta_i > 0 \\ & \forall_{i=1}^{n} : \zeta_i^* > 0 \end{aligned}$$

Let's denote an SVM with such an objective function as $SVM(C, C_+^*, C_-^*, T^*)$. The user inputs C and C^* defined in (2) and two additional parameters: num_+, which is the number of Y^* that will be equal to 1, and ϵ which is a weight to be used later. First, a regular SVM is trained with X, Y, and C, using the objective function of (1), and we find the margin distances of X^*. We take the num_+ most positive margin distances and classify them as 1 in Y^*. Then, we initialize two weights, $C_-^* = \epsilon$ and C_+^* such that $\frac{C_+^*}{C_-^*} = \frac{num_+}{n - num_+}$. Then, we call $SVM(C, C_+^*, C_-^*, Y^*)$, and we greedily find two indices i and j such that $y_j^* = -y_i^*$, $\zeta_i^* > 0$ and $\zeta_j^* > 0$ and $\zeta_i^* + \zeta_j^* > 2$, then we know that flipping y_i^* and y_j^* will reduce (3) and preserve the number of positive num_+ examples. We do this as much as possible, and then set $C_+^* = \max\{2C_+^*, C_+^*\}$ and $C_-^* = \max\{2C_-^*, C_-^*\}$. This increases the importance of the unlabeled data, because its label accuracy should be increasing. Finally, we call $SVM(C, C_+^*, C_-^*, Y^*)$ again, with the updated values of C_+^*, C_-^*, and Y^*, and repeat until $C_-^* = C_+^* = C^*$.

3.2 Spectral Graph Transducer

Another method we are using is a semi-supervised version of k-Nearest Neighbors known as the Spectral Graph Transducer, as described in [J$^+$03]. First, a similarity graph is constructed where the vertices are the data points (both labeled and unlabeled) and have edges weighted by some notion of similarity, like distance in a metric space. And under the 3 goals of achieving low training error, having the transductive algorithm match its inductive counterpart, and achieving approximately the same expected ratio between positive and negative examples in both labeled and unlabeled data, [J$^+$03] reduces the problem of weighted kNN to approximating the unconstrained ratio cut on the similarity graph.

Let L^+ denote the set of labeled positive examples, and let L^- denote the set of labeled negative examples. The unconstrained ratio cut problem can be stated as follows.

$$\arg \max_{(G^+,G^-)} \frac{\text{cut}(G^+, G^-)}{|G^+||G^-|}$$

$$\begin{aligned} \text{subject to} \quad & x \in G^+ \text{ if } x \in L^+ \\ & x \in G^- \text{ if } x \in L^- \end{aligned}$$

Here, $\text{cut}(G^+, G^-)$ denotes the sum of weights of edges between G^+ and G^-. As described in [Dhi01], the problem can be equivalently written as

$$\begin{aligned} & \arg\min_{\vec{z}} \frac{\vec{z}^T L \vec{z}}{\vec{z}^T \vec{z}} \\ \text{subject to} \quad & z_i \in \{\gamma_+, \gamma_-\} \end{aligned}$$

where L is the Laplacian matrix of the graph G, $\gamma_+ = \sqrt{\frac{|L^-|}{|L^+|}}$ and $\gamma_- = -\sqrt{\frac{|L^+|}{|L^-|}}$. This problem, though NP-hard, can be approximated well. The approximation arises in considering the real relaxation of the problem where one minimizes $\vec{z}^T L \vec{z}$ under the constraints that $\vec{z}^T 1 = 0$ and $\vec{z}^T \vec{z} = n$, for which the solution happens to be the second largest eigenvalue of L.

Under the constraint that elements in L^+ must be in G^+ partition and elements in L^- must be in G^- partition, the objective function we must minimize changes to $\vec{z}^T L \vec{z} + c(\vec{z} - \vec{\gamma})^T C(\vec{z} - \vec{\gamma})$, the original objective with an added quadratic penalty, with c being the tradeoff parameter and $\vec{\gamma}$ being a vector with dimension equal to the number of datapoints where γ_i is γ_+ if the i-th datapoint is in L^+, γ_- if the datapoint is in L^- and 0 otherwise.

The algorithm for approximating the minimum of the objective is described in [J$^+$03].

3.3 Semi-supervised Neural Network

Another line of attack explored is implementing a semi-supervised deep neural net. The neural net we constructed in an adaptation the *pseudo-label* algorithm due to [Lee13]. Lee's algorithm consists of a unsupervised phase, a supervised phase, and a semi-supervised phase. The first phase ignores the labels entirely and trains an autoencoder to get a better representation of the data. The second phase uses the labeled data to train the full network using gradient descent and dropout. For the final phase, Lee uses the method of *pseudo-labels* to utilize more fully the unlabeled data. In this method, during each round of gradient descent, the neural net is trained on both kinds of data. The labeled data is treated as normally. For the unlabeled data, a classification is chosen based on most most highly activated output unit for the unlabeled data point, and the accuracy of the neural network on that data point is measured from the inferred classification.

We implemented a modified version of Lee's algorithm. The neural network we use consists of four layers: the input nodes, the hidden layer, and the output nodes. There are 16 input nodes which encode the data in its original form. The autoencoder layer stores a better representation of the data. The hidden layer consists of 12 hidden units. The output layer consists of 26 units, 16 units

which are trained as an autoencoder and 10 units which are trained for classification. Initially all the weights are set to small random values. During the first phase, we train the neural network only on the labeled data, attempting to minimize the squared error of both the autoencoder and the classification units. We use 50 iterations for this stage. For the second phase, we train both the labeled and unlabeled data trying to minimize both the autoencoder and the classification units, where we use the pseudolabels for the unlabeled data to judge the classification accuracy. This lasts for 250 iterations. In both phases, we use the backpropogation algorithm with momentum. We have the following results: [FORMAT CORRECTLY]

0.5% - 75% [UPDATE] 1% - TODO 5% - 90% [UPDATE] 10% - TODO 20% - 90% [UPDATE]

3.4 Error-correcting code for TSVM

To do the 10-way classification with TSVMs, we trained 10 different "one versus rest" classifiers. Another approach is training many classifiers using different kinds of splits and aggregating the results with an error-correcting code to decide on a classification. This approach is inspired by the recent work of [BDM15]. [MORE DETAIL]

3.5 Autoencoder + TSVM

[ACTUALLY DO THIS] An additional aspect we explored is trying different representations of our feature space (besides kernels). The different representation we tried is a compressed one using an autoencoder. We modified the neural network code mentioned above to produce to train an autoencoder on both the labeled and unlabeled training examples. Using the representation given by the hidden units in this autoencoder, we then trained a family of TSVMs to classify this data.

4 Data, Experiments, and Results

So far, we have exclusively utilized the "Pen-Based Recognition of Handwritten Digits Data Set" from [AA98]. The data presented is a normalized time series of a user drawing a digit. Specifically, each input consisted of eight points in $[0, 100]^2$, representing the location of the pen at each step of drawing the digit. Often the originally collected data by the designers had more steps, but the data set was normalized to achieve uniformity of data while still preserving accuracy. The classifier h we seek to train consists is thus a function from $[0, 100]^{16} \to \{0, 1, \ldots, 9\}$. As the goal of this project is semi-supervised learning, we partitioned the data into three groups, labeled training data, unlabeled training data, and testing data. The data was already pre-divided into training and testing batches. We used random sampling to make some fraction of the data unlabeled. Our tests would use 0.5%, 1%, 5%, 10%, 20%, and 50% labeled data to understand the impact of more labeled data on our methods.

We tested the searching algorithm used in [Joa99a] on this data set. We trained 10 different TSVMs, one to distinguish one digit from the rest of the data. To classify a given test point, we found the signed distance from the point to the decision boundary of each TSVM and found the maximum. We used scikit-learn's SVM implementation as the basis of our TSVM. We tried using three different kernels for our TSVM: linear, Gaussian, and sigmoid. The Gaussian and sigmoid kernels caused our TSVMs to predict all test points as -1, achieving 90% accuracy. We believe this is because the two kernels can express functions that are too complicated for the noise and high-dimension of this data set. Our results proved better with the simple linear kernel, and some results are plotted in the tables below:

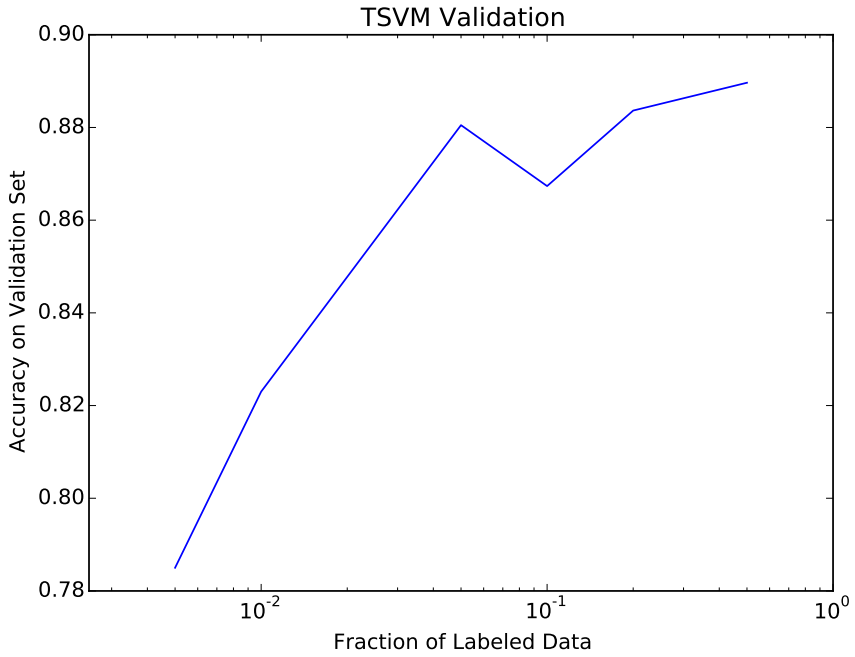
Finally, we produced a noticeable relationship between percent of labeled data and classification accuracy

As can be seen from Table 1, certain pairs of digits were more likely to be confused by the algorithm. For example, a large number of 0's were classified as 8's and a large number of 1's as 5's, and a fair number of 9's as 1's. A likely explanation for why some of the 0's were misclassified as 8 is that some instances of 0 were written with a slash through (like '∅' as opposed to '0').

Table 1: This table represents the accuracy of our TSVM classification algorithm with 20% labeled data. The overall accuracy is 88%. Each column represents the actual digit which was drawn, and each row represents the label given by our algorithm. For example, there were 4 5's which were incorrectly labeled as 1's.

	0	1	2	3	4	5	6	7	8	9
0	311	0	0	0	0	0	0	0	3	0
1	1	284	14	3	1	4	0	34	0	31
2	0	14	347	0	0	0	0	2	1	0
3	0	0	0	329	0	18	0	0	0	4
4	1	1	0	1	326	0	0	13	0	1
5	0	61	0	0	19	277	0	6	20	11
6	8	1	0	0	6	3	333	7	0	0
7	0	2	3	2	0	0	0	294	6	2
8	42	0	0	0	0	3	3	6	306	3
9	0	1	0	1	12	30	0	2	0	284

Figure 1: The following graph plots accuracy of our TSVM classifier based on the fraction of labeled data. We ran the TSVM on 0.5%, 1%, 5%, 10%, 20%, and 50% labeled data. Note that nearly 80% accuracy was achieved even with 0.5% labeled data.



5 Future Plans

[INTEGRATE THESE THINGS]

For future work, we plan to test the Spectral Graph Transducer algorithm described in section 3.2. We have also not yet incorporated the second data set from [LPM$^+$08]. We may end up deciding to not incorporate this data at all, since the data is in a different format and it has more symbols such as Latin letters and Unicode characters and there are fewer samples per character.

Another area of exploration is trying different kernels for the TSVM, perhaps something more complex than a linear kernel and not as advanced as a Gaussian kernel. Doing feature selection could help us find a good kernel. For example, we could change the algorithm of [Joa99a] by not enforcing exactly num_+ examples to be classified as +1. Another approach would be to use the Graph kernel methods of [SK03]. We would also like to try different values for the parameters in the algorithm.

References

[AA98] E Alpaydin and Fevzi Alimoglu. Uci pen-based recognition of handwritten digits data set. 1998.

[BDM15] Maria-Florina Balcan, Travis Dick, and Yishay Mansour. On the geometry of output-code multi-class learning. *CoRR*, abs/1511.03225, 2015.

[BM98] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[BNS05] Misha Belkin, Partha Niyogi, and Vikasy Sindhwani. On manifold regularization. In *AISTATS*, 2005.

[Dhi01] Inderjit S Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–274. ACM, 2001.

[J$^+$03] Thorsten Joachims et al. Transductive learning via spectral graph partitioning. In *ICML*, 2003.

[Joa99a] T Joachims. Transductive inference for text classifcation using support vector machines. In *ICML*, pages 200–209, 1999.

[Joa99b] Thorsten Joachims. Making large scale svm learning practical. Technical report, Universität Dortmund, 1999.

[Lee13] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, 2013.

[Lic13] M. Lichman. UCI machine learning repository, 2013.

[LPM$^+$08] David Llorens, Federico Prat, Andrés Marzal, Juan Miguel Vilar, María José Castro, Juan-Carlos Amengual, Sergio Barrachina, Antonio Castellanos, Salvador España Boquera, JA Gómez, et al. The ujipenchars database: a pen-based database of isolated handwritten characters. In *LREC*, 2008.

[SK03] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003.