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), spectral graph transducers, and neural networks. We exposit our experimental results with these methods, and explore major takeaways of this project.

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). Our algorithms build off of SVMs and neural nets, both supervised models.

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], and we describe his algorithm briefly. 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. Recall that an SVM reduces to the following primal problem:

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

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

Here, we use a more general version of an SVM:

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

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

To implement our TSVM, the user inputs constants C, the weight given to the labeled data, C^*, the weight given to the unlabeled data, 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). We take the num_+ most positive margin distances of X^* and classify them as 1 in Y^*. Then, we initialize two weights, $C_-^* = \epsilon$ and $C_+^* = \epsilon \frac{num_+}{n - num_+}$. Note that this gives more weight to whichever label, $+$ or $-$, should have more instances, and that both weights are small, representing that the accuracy of the unlabeled weights is not very high. Then, we call an SVM with the objective function of (2), using both X and X^* as our input data and Y and Y^* to be as our outputs. This new SVM may have two points, x_i^* and x_j^* in X^*, that have opposite labels yet are both misclassified by the SVM, or one is very misclassified and the other is classified correctly, but in the margin. In this case, we flip the labels of x_i^* and x_j^* because this preserve the number of positive num_+ examples and reduces the objective function of (2). We do this for as many points 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. Then, we repeat the above procedure, training an SVM with the updated values of C_+^*, C_-^*, and Y^*, and repeating until $C_-^* = C_+^* = C^*$.

To classify the digits, we trained ten one-vs-all TSVMs, one for each digit. To classify a new digit, we output whichever digit had the TSVM with the highest positive margin. We feel that the margin of the TSVM roughly corresponds with how confident the TSVM is. We also try error-correcting codes (Section 3.3). We experiment with four kernels: polynomial, Gaussian, linear, and sigmoid.

3.2 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 sum of the squared error of the autoencoder and the classification units, where we weight the autoencoder three times as much as 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. We also weight the autoencoder now four times as much instead of three times as much to emphasize the use of the unlabeled data to assist the autoencoder. This lasts for 250 iterations. In both phases, we use the backpropogation algorithm with momentum as described in the lecture. The gradient descent step size was steadily decreasing for the labeled data while it was steadly increasing for the labeled data.

3.3 Error-correcting code for TSVM

A different approach to training binary TSVMs to classify digits is to train 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].

In this approach, we associate each class (in our case, each class is a digit) to a point in the space $\{-1, +1\}^k$ for some appropriately chosen k. We choose our points as codewords of some error correcting code, which for our purposes means that the representations of any 2 digits in this transformed space are far apart: a digit would look something like $(-1, 1, 1, -1, -1, 1)$. A separate transductive SVM is trained on each entry in the tuple, and classification is performed on a new datapoint x by picking the class that is least Hamming distance away from the classification $(h_1(x), \ldots, h_k(x))$ where the h_i are the separately trained classifiers.

The representations of the digits we chose were the codewords of the (7,4)-Hamming code [MS77]. They have the property that any any two codewords are Hamming distance 3 apart, and are robust up to a single corruption. This means that if the class for digit 2 was $(-1, 1, -1, 1, 1)$ and the classifier on the fourth bit was incorrect leading to $(-1, 1, -1, -1, 1)$, the algorithm would still correctly classify the datapoint as digit 2.

A distinct advantage this might have over the one-vs-all classifier is its robustness if one classifier makes a mistake.

3.4 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.

3.5 Spectral Graph Transducer

Another method we use 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

$$\arg \min_{\vec{z}} \frac{\vec{z}^T L \vec{z}}{\vec{z}^T \vec{z}}$$

$$\text{subject to} \quad z_i \in \{\gamma_+, \gamma_-\}$$

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].

4 Data, Experiments, and Results

We have exclusively utilized the "Pen-Based Recognition of Handwritten Digits Data Set" from [AA98]. We had previously considered supplementing with data from another We have also not yet incorporated the second data set from [LPM$^+$08], but we decided not to incorporate this data since it is in a different format and it has more symbols such as Latin letters and Unicode characters and there are fewer samples per character.

The data presented in [AA98] 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 with two different approaches. In one approach, we trained 10 different TSVMs, one to distinguish one digit from the rest of the data (the approach we have been referring to as one-vs-all approach. To classify a given test point,

Table 1: This table represents the accuracy of our TSVM classification using the error correcting codes approach and the Gaussian kernel with 20% labeled data. The overall accuracy is 96.4%. Each column represents the actual digit which was drawn, and each row represents the label given by our algorithm. For example, there were several digits with multiple instances incorrectly labeled as 1's, whose codewords were relatively close to the codeword of 1.

	0	1	2	3	4	5	6	7	8	9
0	351	0	0	0	0	0	0	0	3	0
1	10	339	1	1	1	12	1	15	5	12
2	0	19	362	0	0	0	0	3	0	0
3	0	0	0	330	0	5	0	0	0	0
4	0	1	0	0	351	0	0	0	0	0
5	0	0	0	0	9	313	1	0	1	0
6	8	1	0	0	6	3	334	1	0	0
7	0	5	1	3	3	5	0	345	0	4
8	2	0	0	0	0	0	0	0	330	1
9	0	0	0	2	0	0	0	2	0	319

we found the signed distance from the point to the decision boundary of each TSVM and found the maximum.

The second approach we used was the one based on error correcting codes, described in section 3.3.

A baseline we had earlier was one-vs-all using a linear kernel, which performed at 88% accuracy on 20% labeled data. Using the polynomial kernel $(x_i \cdot x_j + 3)^5$ brought the test accuracy up to 96%, and using a Gaussian kernel where the kernel function was of the form $e^{-0.8\|x_i - x_j\|^2}$ earned a test accuracy of 97%. The error correcting codes technique performed slightly worse, with a test accuracy of 82% on the linear kernel, 95% on the polynomial kernel and 96% on the Gaussian kernel.

We found it surprising that a straightforward one-vs-all approach performed better than the error-correcting codes approach, since the latter approach was engineered to be robust to classification error.

An approach that might have yielded better results is using a code with greater redundancy such as the Hadamard code [MS77]. This way, the algorithm would be robust to more errors on separate entries of the tuple. Additionally, the way the codewords were assigned to digits was arbitrary and a more careful assignment that put more distant codewords on similar digits (like 1 and 7, or 4 and 9) would decrease the chance of the algorithm confusing similar digits.

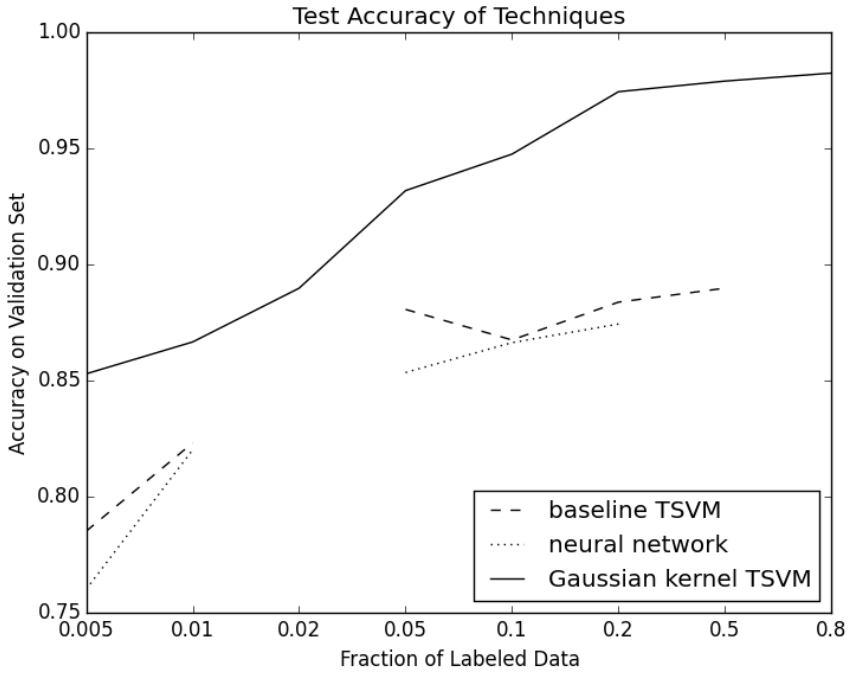
Our implementation of one-vs-all took the margin of classification into account (in particular, it returned the class with maximum margin of classification), but our approach using error correcting codes does not do that, which might be a reason one-vs-all slightly outperforms error correction. This may suggest that having a high margin on a single classifier is more indicative of the correct class than the number of classifiers that correctly classified a point.

The neural network did about as well as the baseline TSVM approach. Also, it is not clear that the unlabeled data assisted much except perhaps prevent overfitting.

Finally, we produced a noticeable relationship between percent of labeled data and classification accuracy [REMAKE PLOT FOR GAUSSIAN KERNEL ON one-vs-all ON 20% LABELED DATA AND (POSSIBLY) ADD CONFUSION MATRIX FOR one-vs-all]

As can be seen in the confusion matrix of the error correcting codes, 5, 7 and 9 were misclassified by 1's and the codewords for these digits that we picked had Hamming distance 3 away from 1, which explains why the confusion was likely.

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%, 2%, 5%, 10%, 20%, 50%, and 80% labeled data. Note as the amount of labeled data increases, it becomes marginally less useful.



5 Future Plans

[INTEGRATE THESE THINGS]

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.

6 Conclusions and Lessons Learned

From completing this project, we have learned a few important ideas about semisupervised learning.

We learned the notion of the bandwidth of a kernel, that is, how certain constant parameters can make a big difference in how well a kernel SVM performs. For example, when we initially tried the Gaussian kernel, it performed poorly since the value c in $e^{-c\|x_i - x_j\|^2}$ was fairly large. On tuning c however, the performance was much better. In particular, before tuning c, the decision boundary looked like a collection of small balls around the datapoints.

When implementing the neural networks, we learned that it is quite difficult to use the unlabeled data without falling into a confirmation bias. Using the autoencoder was quite useful in that regard. It is not clear how much the unlabeled data assisted in finding a better fit. Furthermore, more experimentation could have been done to determine the correct momentum and step sizes.

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.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

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