Mathematical Aspects
of Machine Learning

Report

Digit Recognition with Support Vector Machines

Authors:
Lisa Gaedke-Merzhäuser
Paul Korsmeier
Lisa Mattrisch
Vanessa Schreck

1 Problem Statement and Introduction to SVM

Our main goal is to correctly identify handwritten digits based on the MNIST ("Modified National Institute of Standards and Technology") data set. This data set consists of 42,000 gray-scale images. Each image is 28 pixels in height and 28 pixels in width. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel (kag).

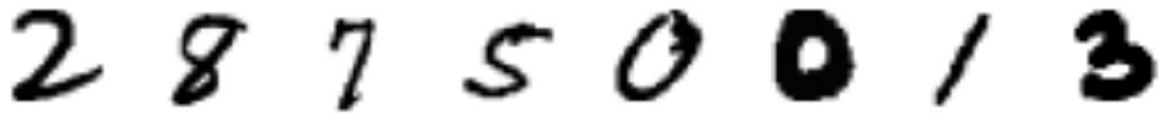

Figure 1: Visualization of eight of these images

Formally speaking, we have points $\{x_i\}_{i=1}^{l} \subset \mathbb{R}^{28^2}$ with respective labels from $\{0, 1, \ldots, 9\}$. Our goal is now to find a classification function f that gives for each of these points a prediction $f(x) \in \{0, 1, \ldots, 9\}$ which ideally matches the actual label.

There are several ways to achieve this goal, one of which are the concept of kernelized support vector machines (SVM). The main idea behind this algorithm is to not only separate the data according to their labels, but also do so in a way that is as reasonable as possible, i.e. we want to maximize the distance of the data to the decision boundary. We first transfer the data with a feature map Φ into a suitable feature space and look for a linear separation there.

From this notion, one particular problem arises: The SVM is a binary classifier, i.e. it can only handle data that is divided into two classes. We therefore decided to compare different ways to deal with this obstacle, generalizing the application of SVMs to multi-class classification problems (cf. Section 3). The methods that we considered all combine several instances of the SVM in its original form, classifying according to labels $\{\pm 1\}$.

We can summarize this binary classification problem in the objective of finding a decision function of the form

$$f(x) = \operatorname{sgn}\left(w^T \Phi(x) + b\right).$$

1.1 Binary Support Vector Machine

Approaching binary support vector machines from a more technical side, we find that they aim to find the maximum margin hyperplane in the feature space, i.e. the goal is to maximize the margin while softly penalizing points that lie on the wrong side of the boundary or inside the margin (Bis06). Dualizing this optimization problem, we obtain the following equivalent quadratic program:

$$\begin{aligned} \text{minimize} \quad & d(\alpha) := \tfrac{1}{2}\alpha^T Q \alpha - 1^T \alpha \\ \text{s.t.} \quad & 0 \leq \alpha \leq C \quad \text{and} \quad y^T \alpha = 0, \end{aligned} \tag{1.1}$$

where α is the dual variable, $q_{ij} = y_i y_j k(x_i, x_j)$, k the kernel function and $C = 1/2\lambda l$ for the penalty term λ of the soft margin SVM. After finding the optimal solution α^* of this

program, we can formulate the resulting decision function as

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i^* y_i k(x_i, x_j) + b\right).$$

Note that only data points x_i with $\alpha_i \neq 0$ appear in the decision function. Those points are called *support vectors*.
By analysing primal, dual and the corresponding KKT conditions, we find that any x_i with $\alpha_i = C$ was correctly classified and lies exactly on the margin boundary, and any x_i with $0 < \alpha_i < C$ lies either inside the margin or on the wrong side of the margin. Additionally, any data points x_i on the (correct) margin boundary must satisfy $y_i f(x_i) = 1$. We can use this property to find the parameter b (Bis06).

2 Solving the optimization problem with SMO

Let α be some feasible variable for Problem (1.1). Defining

$$F_i(\alpha) := y_i(\partial_i d)(\alpha) = \sum_{i=1}^{l} \alpha_j y_j k(x_i, x_j) - y_i \quad \text{for} \quad i = 1, \ldots, l, \tag{2.1}$$

by careful computation we find that the KKT optimality conditions for a solution of Problem (1.1) (which are both necessary and sufficient, since Q is spsd), are equivalent to the – much simpler looking – pairwise condition

$$b_{up}(\alpha) := \min_{i \in I_{up}(\alpha)} F_i(\alpha) \geq \max_{j \in I_{low}(\alpha)} F_j(\alpha) =: b_{low}(\alpha), \tag{2.2}$$

where $I_{up}(\alpha)$ and $I_{low}(\alpha)$ are subsets of the index set $\{1, \ldots, l\}$ defined by

$$I_{up}(\alpha) := \{i \mid \alpha_i < C \text{ and } y_i = 1 \text{ or } \alpha_i > 0 \text{ and } y_i = -1\} \tag{2.3}$$
$$I_{low}(\alpha) := \{j \mid \alpha_j < C \text{ and } y_j = -1 \text{ or } \alpha_j > 0 \text{ and } y_j = 1\}. \tag{2.4}$$

Any pair $(i, j) \in I_{up}(\alpha) \times I_{low}(\alpha)$ with $F_i(\alpha) < F_j(\alpha)$ is thus called a *violating pair* and an objective equivalent to solving Problem (1.1) is to change α so as to remove all such violating pairs. Since a priori we do not know if the solution α^* fulfils (2.2) strictly or not, we define, for some small tolerance $\tau > 0$, a τ-violating pair as some $(i, j) \in I_{up}(\alpha) \times I_{low}(\alpha)$ which satisfies $F_i(\alpha) < F_j(\alpha) - \tau$ and require that all τ-violating pairs be removed, or equivalently,

$$b_{up}(\alpha) \geq b_{low}(\alpha) - \tau. \tag{2.5}$$

It holds that any algorithm of the following form terminates after finitely many steps:

Algorithm 1 (General SMO type algorithm)**.** Let $\tau > 0$. Initialize $k = 0$ and $\alpha = 0$ and generate iterates α^k, $k \in \mathbb{N}$, as follows:

1. If α^k satisfies (2.5), stop. Else choose a τ-violating pair $(i, j) \in I_{up}(\alpha^k) \times I_{low}(\alpha^k)$.
2. Minimize d varying only α_i^k and α_j^k, leaving α_n^k fixed for $n \notin \{i, j\}$ and respecting the constraints of Problem (1.1) to obtain α^{new}.
3. Set $k := k + 1$, $\alpha^k := \alpha^{\text{new}}$ and go to Step 1.

Platt's original SMO algorithm (actually, we are referring to "Modification 1" by (**?**)[KEERTHI ET AL. SVM, PAPER]), although ground-breaking, has two weaknesses. Firstly, its pseudocode description is quite complex, making it hard to judge whether or not one has implemented it as intended by its originators. Secondly, by trying to avoid computational effort in the *while* steps, it actually runs much longer than all other algorithms we have implemented or tested.

In short, Platt's SMO tries first to ensure that

$$b_{up,I_0}(\alpha) := \max_{i \in I_0(\alpha)} F_i(\alpha) \geq \min_{j \in I_0} F_j(\alpha) =: b_{low,I_0}(\alpha),$$

where $I_0(\alpha) := \{i \mid 0 < \alpha_i < C\}$ is the index set of "interior" α_is. A cache of only the F_i for $i \in I_0$ is kept until this is achieved. Then all $i \in \{0, \dots l\}$ are examined, and the cache of correctly stored F_is (along with the indices $\widetilde{i_{up}}$ and $\widetilde{i_{low}}$ indicating where the so far most extreme F_i occur) is extended as long as an α_j is found such that $(j, \widetilde{i_{low}})$ or $(\widetilde{i_{up}}, j)$ is violating, depending on whether $j \in I_{up}$ or $j \in I_{low}$. Therefore, the algorithm does not resolve a *maximally* violating pair, by which we mean some

$$i_{up}, i_{low}) \in \operatorname{argmin}_{i \in I_{up}} F_i(\alpha) \times \operatorname{argmax}_{j \in I_{low}} F_j(\alpha).$$

To look up such a pair is the strategy of the "Working Set Selection 1" (WSS1) approach in [PAPER VON FAN, CHEN ...] and a rewarding investment, see benchmarking section.

Since, actually, for a pair (i, j), the term $F_i(\alpha) - F_j(\alpha)$ is the derivative at α of the 1D problem in Step 2 of Algorithm 1, WSS1 corresponds to a steepest descent approach aiming for a substantial decrease in d. Since this uses only first order information, a second "Working Set Selection 2" (WSS2) strategy is proposed in [PAPER VON FAN, CHEN ...], which (at relatively low extra cost) employs second order information to choose a violating pair almost optimally, optimally meaning maximizing the descent of d in the SMO step.

3 Multiclass Classification Problem

Support Vector Machines are binary classifiers, meaning that they only have two possible output labels. Often one chooses them to be 1 and -1. Our problem however was to classify unknown data into one out of 10 different categories. Since there is no direct way to change one support vector machine so that it can choose from more than two labels we had to find a way to combine several SVMs, who together would be able to do so. There are a number of well known strategies tackling this issue. Let us first look at a very intuitive but also primitive approach called One-vs-All classification. Using this method one trains as many SVMs as one has classes, so 10 in our case, and sets labels such that the i-th SVM has all training data with label i set to one and everything else to -1. When trying to predict a new label one simply looks for a classifier that gave this data point the label one. Unfortunately this approach has many issues. First of all new data points are generally not uniquely classified. Therefore it is necessary to add some sort of confidence score which label is considered to be most likely out of the possible ones. Finding good ways of assigning reliable confidence scores is often not an easy task. We decided to resolve this issue by computing the barycenters of each class and when ambiguities

arose chose the label of the closest barycenter. This simple strategy notably improved our results and we decided on not devoting any more time on a different strategy because the One-vs-All classification has another major drawback. The number of training points for each classifier being assigned positive and negative labels is very unbalanced, in our case on average 1 to 9. Therefore one is likely to get shifted margins to what one would normally perceive to be correct. The penalty term will lead each classifier to give less importance to the positives than the negatives simply because they are less in numbers and can be outweighed by the others.

Bild hier oder zu unwichtig?

Another common intuitive approach would be the One-vs-One classification. Here during training one only ever considers the data corresponding to two classes and sets one label to 1 and the other one to -1 for every possible pair. In our case this would mean training 45 (the number arises from the binomial coefficient of 10 choose 2) different SVMs however with data sets of a fifth of the original size. Because the issue of how to deal with ambiguities gets even worse we decided not to implement this approach and instead focus our attention on a different strategy that seemed like a much better idea to us and indeed as the next chapter will reveal, yielded very good results:

3.1 Error Correcting Output Codes

They are easiest to explain considering a concrete example. We trained 15 different classifiers f_i. The rows of the table show what class is assigned what label depending on each classifier. For example f_0 assigns 1's to all even numbers and -1's to all odd numbers. This way each class has a string or codeword of 1's and -1's it corresponds to.

Table 1: Error Correcting Output Codes

Class	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	-1	-1	-1	-1	1	-1	1	-1	-1	1	1	-1	1
1	-1	-1	1	1	1	1	-1	1	-1	1	1	-1	-1	1	-1
2	1	-1	-1	1	-1	-1	-1	1	1	1	1	-1	1	-1	1
3	-1	-1	1	1	-1	1	1	1	-1	-1	-1	-1	1	-1	1
4	1	1	1	-1	1	-1	1	1	-1	-1	1	1	-1	-1	1
5	-1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	1
6	1	-1	1	1	1	-1	-1	-1	-1	1	-1	1	-1	-1	1
7	-1	-1	-1	1	1	1	1	-1	1	-1	1	1	-1	-1	1
8	1	1	-1	1	-1	1	1	-1	-1	1	-1	-1	-1	1	1
9	-1	1	1	1	-1	-1	-1	-1	1	-1	1	-1	-1	1	1

The codewords are chosen such that their Hamming distance (i.e. the number of entries where they differ) is maximized. In our case each codeword has a Hamming distance of at least six to any of the others strings. Now when one wants to label an unknown data point each of the classifiers assigns it a label and one gets an output string of 15 digits. We classify the data point according the codeword it has the least Hamming distance to. The name Error Correcting Output Codes is derived from the fact that for example any three classifiers can misclassify a data point and we will still get the correct result. Depending on the difficulty of the problem one could choose more or less classifiers and

hece in-or decrease the Hamming distance of the set of codewords. We took this idea as well as the codewords given above from (DB95).

4 Results, Difficulties & Conclusions

When choosing ... many training points and .... many testing points our algorithm was correct in ... of the cases. Include things here...?

Linear and Gaussian.

In general ECOC?

Where do we bring in cross validation?

General Conclusions? Difficulties?

For One-vs-All linear almost (<5 %) no correct result, barycenters improved things a lot (maybe take out of table if its too bad...)

One-vs-All Gauss: a lot better, I think especially more training data was helpful

ECOC

Table for overview of results

Each time we tested with ... 1000 (!!!!!!!!!!!) test points. The given training data set that we had was very large, so we always had data to test our results that was not used for training.

Talk about average compute times. Or did Paul already say something about that beforehand?

—–> Sorry, haven't really tracked compute times consistently. Will, although, compute ecoc Gauss for 10000 training points again with what turned out (surprisingly) to be our fastest SMO (Namely WSS1, not WSS2 as I assumed.) Will be around 2 hours or so. Will give the exact figure. Then we can say at least how long the longest job took. Is that, combined with an extra benchmark section only focused on the comparison of the 5 SMOs applied to only the first of the 15 ECOC classifiers, both kernels, enough?

Talk about the 1500 issue or better not? —-> No. I resolved the issue and I suppose it was a strange coincidence that the problem occurred specifically at 1500 training points. Will talk about the cause of this problem and its resolution in the SMO section.

Table 2: Overview Results

no of training points classifier	One-vs-All uniquely classfied correctly linear	One-vs-All w/ barycenters linear	One-vs-All uniquely classfied correctly Gaussian	One-vs-All w/ barycenters Gaussian	ECOC linear	ECOC Gaussian
500	659	741	754	833	747	874
1000	682	750	843	890	787	927
2000	702	764	898	919	778	943
5000	700	738	889	916	820	952
10000	646	675	880	906		954

References

[Bis06] Bishop, Christopher M.: *Pattern Recognition and Machine Learning* -. 1st ed. 2006. Corr. 2nd printing 2011. Berlin, Heidelberg : Springer, 2006. – ISBN 978–0–387–31073–2

[DB95] Dietterich, Thomas G. ; Bakiri, Ghulum: Solving multiclass learning problems via error-correcting output codes. In: *Journal of artificial intelligence research* 2 (1995), S. 263–286

[kag] `https://www.kaggle.com/c/digit-recognizer/data`