Project on Support Vector Machines

Natasa Farmaki - DS3517018

Stratos Gounidellis - DS3517005

Course: Optimization Techniques

Instructor: E. Markakis

Athens, June 2018

Support Vector Machines Overview

Support Vector Machines is a supervised machine learning algorithm which can be used for either classification or regression tasks. Indeed, it is considered to be one of the best supervised learning algorithms. Given a set of training examples, each of them assigned to a class (either to -1 or to 1 in the simple case) we aim to predict the class of the test examples in a classification task. Using the SVM machine learning algorithm we aim to compute that hyperplane that separates the two classes better.
Support vectors are simply the closest vectors, from each class, to the classifier. Moreover, apart from linear classification Support Vector Machines can be used for non-linear classification. In order to achieve that we use the kernel trick, i.e. we implicitly map the input into high-dimensional feature spaces. Examples of kernel functions are:

1. Radial basis functions (for appropriate values of σ)
2. Polynomial (for appropriate values of q)
3. Hyperbolic tangent (for appropriate values of β and γ)

Sequential Minimal Optimization Overview

Sequential Minimal Optimization is an algorithm invented by John Platt in 1998 at Microsoft Research. It was accepted with a lot of excitement as till then training of SVMs required the use of expensive and relatively slow quadratic programming software.

More specifically, training a support vector machine requires solving a very large quadratic programming optimization problem. However, according to the SMO approach that very large problem is broken into smaller subproblems that are tractable and can be solved analytically. That approach is memory efficient as there is no need to store the constraint matrix. Therefore, SMO is capable of handling large training sets while it is faster for linear SVMs and sparse data sets. Indeed, according to Platt's paper SMO can be up to 1000 times faster than the chunking algorithm on real-world sparse data sets.

When implementing the algorithm several decisions should be taken, such as:

- How to pick the pair of variables that will be examined in each iteration,
- How to pick an initial solution,
- How to set the termination criterion.

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \end{aligned}$$

SMO algorithm is used to solve the dual problem defining SVM classifier.The SMO algorithm solves this problem in O(n^3) in the worst case.

Data Preprocessing

In order to test the implementation of the algorithm gisette dataset was used. This dataset is free and can be accessed on this link.

The dataset was split into three parts. The training set containing 2748 observations, the development set containing 1649 observations and the test set containing 1100 observations. The training set is used to train the Lagrange multipliers(alphas), the threshold(beta) and the weights for the linear SVM. The development set is used to train the hyperparameters C for the linear SVM and C, sigma(variance) for the case of gaussian kernel. Last but not least, the test set is used to evaluate the classifier using multiple metrics (accuracy, F1-score etc.).

Algorithm description

The SMO algorithm was implemented using Platt's paper. The algorithm first picks a pair of variables that will be examined and optimized in each iteration, one of which must violate KKT conditions in order to use Osuna's Theorem and guarantee convergence. The fact that two Lagrange multipliers need to be optimized at each step stems from the fact that we have a linear constraint to satisfy.

Picking Lagrange multipliers to optimize

More specifically, to choose the first variable we find all the examples on the training set that violate KKT conditions and then pick one that is also a non-bound point. In generally the algorithm optimizes the non-bound points to obey KKT conditions until the whole training set is optimized (i.e. obeys the KKT conditions within a bound ε). After finding the first variable to optimize in the current iteration we pick the second variable (Lagrange multiplier).

The choice corresponds to the maximum size of the step taken during joint optimization. For that purpose, we use the error cache that stores $E = u_i - y_i$ for each observation. Then we calculate $|E1 - E2|$ for each combination of the first Lagrange multiplier chosen and the non-bound points. To maximize the step, we pick the non-bound point that has the maximum $|E1-E2|$. More specifically, if E1 is negative then we pick the example with the maximum error E, while if E1 is negative we pick the example with minimum error E.

In case SMO cannot make a positive step with that heuristic a hierarchy of heuristics is utilized. If the above heuristic fails, then we iterate over all the non-bound examples and we pick an example that makes positive progress. If again no positive progress is achieved, then we iterate over all the examples until we find an example that achieves positive progress. In both iterations we start from a random point, so we do not make a biased choice. If both heuristics fail, which is a degenerate case, we select another example as first example and start again from the first heuristic.

Calculating Error Cache

It is observed that the method of calculating E is crucial to the speed of the algorithm. The obvious choice of re-calculating all the error cache at each iteration is too time consuming and thus we need to find a more efficient way to eliminate the two nested loops.

The idea of that optimization is that instead of recalculating the error for every non-bound example in the training set (as Platt indicates) we can just update the cache adding the output(u) using the Lagrange multipliers that changed in the current step and subtract the new threshold b from the previous one. Then we can set the error of the two optimized Lagrange multipliers to zero and we are done. This calculation needs only one loop over the non-bound training examples and thus need O(k) time where k=non-bound points.

Kernels

After choosing the Lagrange multipliers we solve a system of one inequality and one equality constraint as described in Platt's paper. In the case of the linear SVM the output function is:

u = wx-b. But if the classes are not linearly separable this classifier will achieve a poor performance. To separate non-linearly separable classes we need to use kernels. The output of the algorithm after using kernels is:

$$u = \sum_{j=1}^{N} y_j \alpha_j K(\vec{x}_j, \vec{x}) - b$$

In both cases after we calculate u we classify u > 0 in class 1 and u < 0 in class -1. In the degenerate (or in case of underflow) case where u = 0 we choose randomly between the two classes.

In this implementation appart from the linear kernel, we used the gaussian kernel to classify the dataset. We observed that in the simple implementation of calculating the kernel function every time was too time consuming. Thus, we stored the output of the gaussian kernel function in a matrix to avoid calculating the kernel at each iteration. It turned out that this change made the algorithm significantly faster. In order to improve memory efficiency we could have stored the matrix in compressed sparse form.

Terminating condition

The outer loop of the algorithm iterates over the non-bound examples until all those examples satisfy the Karush–Kuhn–Tucker (KKT) conditions within a specific tolerance level. After the outer loop iterates again over all the examples in order to ensure that all the examples satisfy the KKT conditions within that tolerance. Finally, the algorithm terminates when KKT conditions are satisfied for all observations within a specific tolerance. In our implementation, tolerance is predefined to 10^{-3}; however, it is a parameter that can be tuned.

Initial Values

A last issue to tackle during the implementation is the initial values of the vector of Lagrange multipliers alpha, of threshold beta and of the error cache.

- As the linear constraint of the problem $\Sigma y_i * a_i = 0$ must be satisfied we initialize all **alphas** with zero.
- Threshold **beta** is arbitrarily set to 0.
- The **error cache** is thus, set to minus target. The formula for calculating error cache is $E = u_i - y_i$ where target is the real classes of each observation. As the output u is equal to zero when corresponding alpha and beta are zero, E will eventually be equal to minus target.
- The **initial weights** for the linear SVM are also set to 0. The weights are calculates as $\Sigma a_i * y_i * x_i$. Since alphas are set to 0 then from the aforementioned formula it is obvious that initial weights are also 0.

Evaluation Metrics

In order to evaluate the results of the multiple classifiers and therefore compare their performance we need to implement appropriate measures. The metrics that can holistically evaluate performance are accuracy, recall, precision and F1-score (produced by both recall and precision).

Accuracy is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage. Precision is the number of True Positives divided by the number of True Positives and False Positives.

Precision can be thought of as a measure of a classifier's exactness. A low precision can also indicate a large number of False Positives. Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate.

Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives.

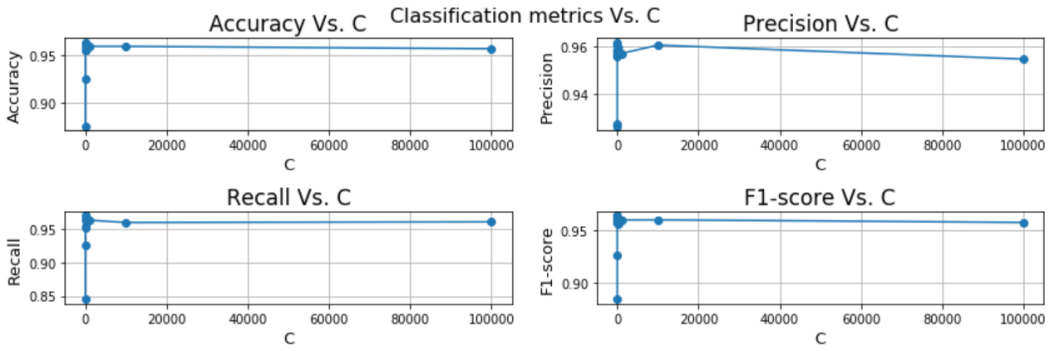
F1-score is the 2*((precision*recall) / (precision + recall)). Put another way, the F1 score conveys the balance between the precision and the recall.

Experimental Results

Baseline Classifier

To get a clear picture of a classifier's performance, it is common to compare its accuracy or error rate to those of a simplistic "baseline" approach. In our case, we used the simplest classifier. Therefore, the simplest classifier would just classify all examples in the most common class of the training set. The accuracy of the baseline classifier will be compared with the accuracy of the SVM classifier. The accuracy of the baseline classifier in the test set is 0.498.

SVM with Linear Kernel



Best accuracy is achieved for C = 10 and it is 0.964 in the development set. In the test set the accuracy of the SVM with Linear Kernel with C = 10 is 0.961.

SVM with Gaussian Kernel

Best accuracy is achieved for C = 100000 and sigma = 10000 and it is 0.965 in the development set. In the test set the accuracy of the SVM with Gaussian Kernel with C = 100000 and sigma = 10000 is 0.967

Conclusions

SVM classifier outperforms the baseline classifier. The data seem to be linearly separable as the linear SVM performs well in comparison to the SVM with the Gaussian kernel.