[AIML2015] Homework 2: Support Vector Machines and Model Selection

Francesco Cucari - 1535743

December 19, 2015

1 Dataset Description

The MNIST dataset is composed of about 70000 images divided into 10 classes of handwritten digits. The digits, from 0 to 9, have been size-normalized and centered in a fixed-size image. An example of MNIST dataset samples is shown below.

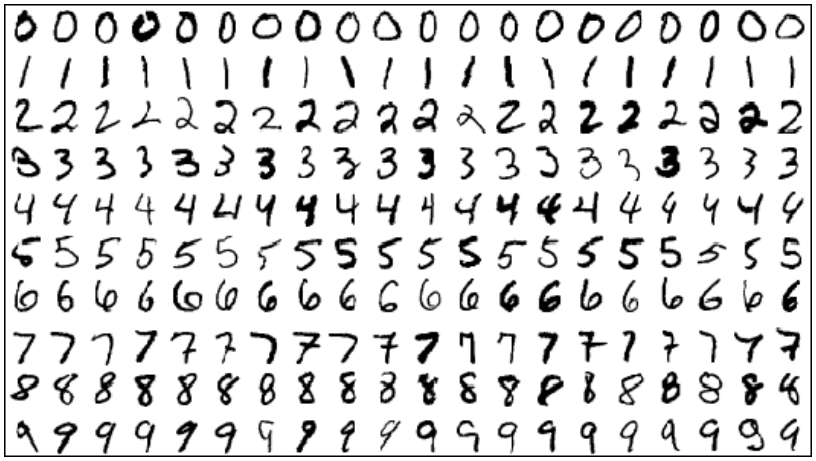

Figure 1: MNIST Dataset

2 Training Linear Support Vector Machine

SVM (Support Vector Machine) is a useful technique for data classification. Given a training sample, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between "positive" and "negative" examples is maximized. Maximizing the margin of separation between binary classes is equivalent to minimizing the Euclidean norm of the weigth vector w.

The margin of separation between classes is said to be *soft* if a data point falls inside the region of separation, but on the correct side of the decision surface or if it falls on the wrong side of the decision surface. In the first case we have correct classification, but misclassification in the second. The solution is to introduce slack variables ξ that measures deviation from the ideal condition.

Now, the goal is to minimize the amount of slack, in order to find a hyperplane with minimum misclassification rate. In other words, we want to maximize the margin and minimize the number of errors that I can tolerate. For this reason, we introduce the parameter *C*. C is a regularization parameter that controls the trade off between the achieving a low training error and a low testing error that is the ability to generalize the classifier to unseen data. Large value of C favor solutions with few misclassification errors. But if we increase C too much we risk losing the generalization properties of the classifier, because it will try to fit as best as possible all the training points (including the possible errors of your dataset). In addition, a large C usually increases the time needed for training. Small value of C denote a preference towards low-complexity solutions.

So, we have to find a C that keeps the training error small, but also generalizes well. There are several methods to find the best possible C. A possible method is to divide the examples in three set: *training*, *validation*, *testing*. The learning of the model proceeds on the training set. Validation set is usually used for parameter selection and to avoid overfitting. When the evaluation seems to be successful, final evaluation can be done on the test set. In our assignment, we split selected data into the training, validation and testing set as 50%, 20% and 30%.

However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

I execute code 4 times, varying parameter C in the range of my choice. These are the accuracies on the validation set against every choice of C:

	C=0.001	C=0.01	C=0.1	C=1	C=2	C=10	C=50
Test1	0.5507	0.8061	0.9835	0.9901	0.9901	0.**9904**	0.9904
Test2	0.5540	0.7775	0.9805	0.9914	0.9911	**0.9917**	0.9917
Test3	0.5576	0.7804	0.9789	0.9907	0.9911	0.9914	**0.9920**
Test4	0.5682	0.7722	0.9673	0.9868	**0.9871**	0.9868	0.9871

The Figures 2-3-4-5 are the plots of accuracies for every choice of C.

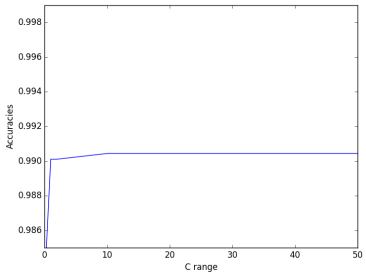

Figure 2: Plot Test1

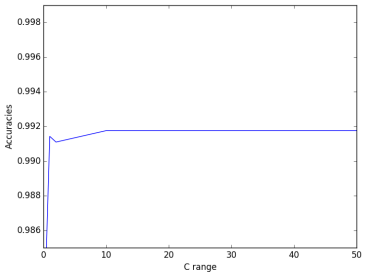

Figure 3: Plot Test2

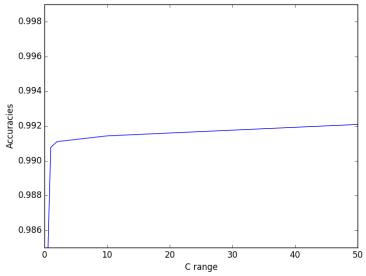

Figure 4: Plot Test3

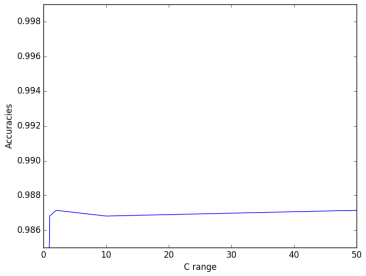

Figure 5: Plot Test4

For the final classifier I choose the smallest parameter C for which one has the highest accuracy. The best accuracies are bold in the table.
After choosing the best C, I train linear binary SVM once again, setting the best C and I test the classifier on the testing set.

In the first test, the accuracy on testing set with best C (10.0) is 0.9909. In the second, the accuracy is 0.9923. In the third, the accuracy is 0.9925 and in the fourth test it is 0.9894.
It is clearly seen that choosing the best C the accuracy on testing set is higher than the one on validation set. This is a very useful result, because it shows that we don't have an overfit model. A model is overfit when it has learned particulars that help it performs better in the training data that are not applicable to the larger data population and therefore results in worse performance.

Finally, I choose **C=50** as the **best C** because the classifier, fitted with it, has the best accuracy on testing set.

3 Training Multiclass Non-Linear SVM

3.1 Why it is important to do stratified splitting?

After the selection of examples which belongs only to classes 0,1,2,3,4, I standardized, shuffled and splitted data into the training and testing set as 50/50% in a stratified way. The *stratified splitting* is important in order to preserve the proportion of the data. In fact when you select a subset of elements from a population, it is important that the selected group is a representative of the population, and is not biased in a systematic manner.
A stratified sample is obtained by taking samples from each stratum or sub-group of a population. When we sample a population with several strata (groups), we generally require that the proportion of each stratum in the sample should be the same as in the population. Some reasons for using stratified splitting are:

- if measurements within strata have lower standard deviation, stratification gives smaller error in estimation;
- it improves the representation of particular strata within the population, as well as ensuring that these strata are not over-represented.

In my code, I use *StratifiedShuffleSplit* (from *sklearn.cross_validation*) that provides train/test indices to split data in training and testing sets. This cross-validation object returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class.

3.2 Multiclass classification through One-Vs-All

A multiclass classification problem can be handled breaking down the K-class classification problem into K separate binary classification subproblems. So, the idea is to solve 5 binary classification subproblems. We assign a binary classifier to any $y \in (0, .., 4)$. Labels of y are treated as positive labels and all of $(0, .., 4) \backslash \{y\}$ as negatives.
For example, I create a new sort of fake training set where classes 1,2,3,4 get assigned to the negative class, and class 0 gets assigned to the positive class. Then, I fit a (binary) classifier. This is the case of class 0 vs All. Then, I repeat this procedure for all the other classes.

For each binary classifier, I have a vector of margin values of classifier. In fact, the method *decision_function(X)* returns a vector of margins, that indicates the distance of each sample from the hyperplane generated by the classifier.

Then, I create a matrix where each row contains the margins of binary classifier of class i *vs all* (with $i \in 0, 1, 2, 3, 4$). By this matrix, I can get a multiclass classifier: for each row (sample) I consider the class that is associated with the highest margin. The higher the margin, the more likely it is correct classification. If the margin is close to 0 then the classification could be not good (the example could be misclassified). If all margins are negative, this means that each classifier "rejects" the sample as belonging to the considered class,

but I consider the highest margin (less negative in module) because it means that the distance from hyperplane is lower and thus is more likely that the classifier has made a mistake.

A SVM classifier equipped with an RBF kernel has at least two hyperparameters that need to be tuned for good performance on unseen data: the regularization constant C and the kernel hyperparameter gamma. In order to have the best performance, I need to tune C and gamma values for each binary classifier. For this purpose, I can use *Grid Search*. Grid Search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. In my code, I use these values:

$$C = [0.01, 0.1, 1.0, 2.0, 10.0, 20.0, 100.0]$$
$$gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]$$

Grid search then trains an SVM with each pair (C, gamma) in the Cartesian product of these two sets and it evaluates their performance by some metric, typically measured by cross-validation on the training set.

So, I apply Grid Search to each binary classifier to tune C and gamma values.
Finally, I compute the accuracy of classifier on the testing set, using the results predicted by the matrix of margins and the value of accuracy is the following:

Accuracy	91.1%