Support Vector Machines and Ensemble Methods

Algorithmic Data Analysis – Coding Assignment 1

Giulia Ortolani

January 29, 2024

Resources: No resources
Collaborations: No collaborations

1 Task 1

In this task I'm implementing the linear SVM algorithm with hard-margin and soft-margin variants. Then, I'm applying the SVM with hard-margin to the irisSV dataset and the SVM with soft-margin to the irisVV dataset.

1.1 SVM with hard-margin – irisSV dataset

After dividing the irisSV dataset into training (4/5) and test (1/5) subsets, I've trained the hard-margin SVM on the training subset and applied the resulting model to the test subset. Figure 1 shows the plot of the hyperplane with the support vectors highlighted as stars. The blue instances indicate the iris setosa species, while the red instances are the iris versicolor (positive class).

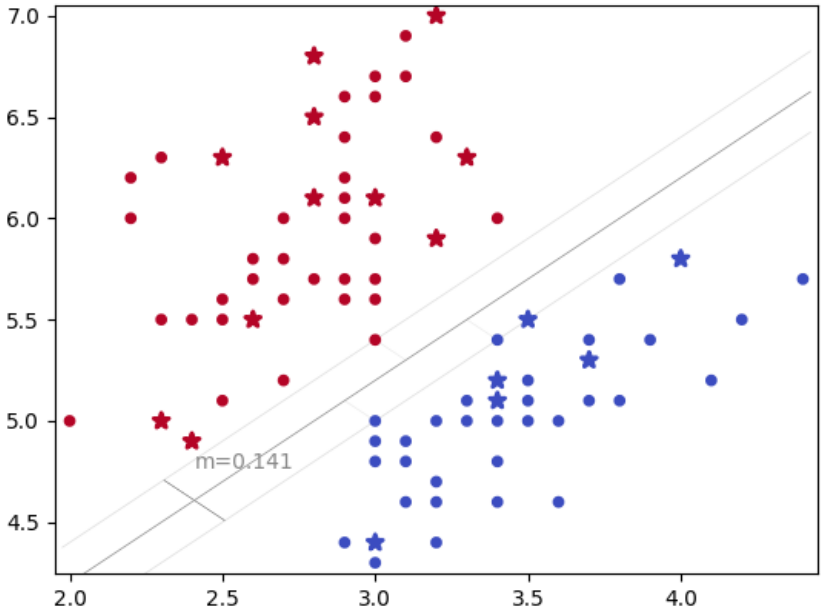

Figure 1: SVM with hard-margin – irisSV dataset

1.1.1 Confusion matrix

Actual / Predicted	Iris setosa	Iris versicolor
Iris setosa	6	0
Iris versicolor	0	11

1.1.2 Accuracy, recall and precision

Recall:	1.0
Precision:	1.0
Accuracy:	1.0

1.1.3 Equation of the separating hyperplane

$$\begin{bmatrix} 5 & -5 \end{bmatrix} \cdot x^T - 10.99 = 0$$

1.2 SVM with soft-margin – irisVV dataset

After dividing the irisVV dataset into training (4/5) and test (1/5) subsets, I've trained the soft-margin SVM on the training subset setting $c = 2$ and applied the resulting model to the test subset.

Figure 2 shows the plot of the hyperplane with the support vectors highlighted as stars. The blue instances indicate the iris virginica species, while the red instances are the iris versicolor (positive class).

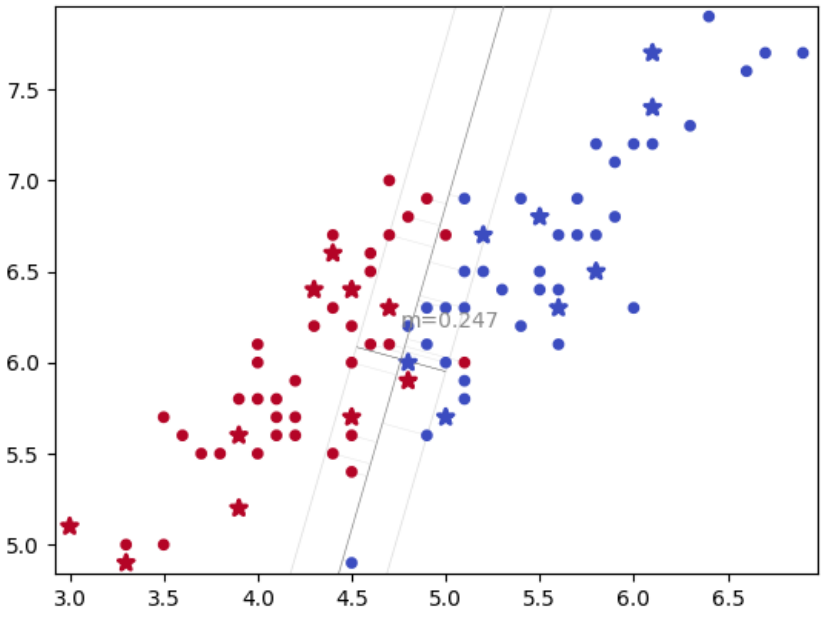

Figure 2: SVM with soft-margin – irisVV dataset

1.2.1 Confusion matrix

Actual / Predicted	Iris virginica	Iris versicolor
Iris virginica	8	1
Iris versicolor	0	9

1.2.2 Accuracy, recall and precision

Recall:	0.9
Precision:	1.0
Accuracy:	0.94

1.2.3 Equation of the separating hyperplane

$$\begin{bmatrix}1.1 & -3.9\end{bmatrix} \cdot x^T - 11.96 = 0$$

2 Task 2

Here I run an evaluation of the soft-margin SVM on the irisVV dataset with 5 fold cross-validation (10 rounds).

	Round									
Fold / Accuracy	1	2	3	4	5	6	7	8	9	10
1/5	0.88	1.0	0.94	0.88	0.88	0.94	0.94	1.0	0.82	1.0
2/5	1.0	0.94	1.0	0.94	0.88	0.94	0.94	1.0	1.0	1.0
3/5	0.88	0.82	0.94	0.88	0.94	0.94	0.94	0.94	0.94	0.94
4/5	0.94	0.82	0.94	1.0	1.0	0.94	0.94	0.88	0.94	0.94
5/5	1.0	1.0	0.88	0.94	1.0	1.0	0.94	1.0	0.94	0.76
Mean Accuracy	0.94	0.92	0.94	0.93	0.94	0.95	0.94	0.96	0.93	0.93
Variance of Accuracy	0.0028	0.0064	0.0014	0.0019	0.0028	0.0006	0.0	0.0022	0.0033	0.0075

Table 1: Soft-Margin SVM Evaluation Results

Moreover, averaging the accuracy and the variance obtained for each round we can obtain:

Overall Mean Accuracy:	0.939
Overall Variance of Accuracy:	0.0029

3 Task 3

These are the results of applying AdaBoost with different value for β:

Beta	**1**	**0.8**	**0.5**	**0.2**	**0**
Training	**Error rate**	**Error rate**	**Error rate**	**Error rate**	**Error rate**
Classifier n.1	0.21875	0.21375	0.22375	0.22375	0.2225
Classifier n.2	0.2125	0.22125	0.21875	0.21875	0.225
Classifier n.3	0.2175	0.215	0.2175	0.21875	0.21625
Classifier n.4	0.21125	0.21125	0.22125	0.21125	0.21375
Classifier n.5	0.21625	0.21625	0.22125	0.21375	0.22125
Classifier n.6	0.21625	0.21625	0.21	0.2075	0.2275
Classifier n.7	0.2175	0.22375	0.21	0.21125	0.22625
Classifier n.8	0.21375	0.21375	0.21	0.21875	0.21875
Classifier n.9	0.21625	0.21875	0.20875	0.20875	0.2325
Classifier n.10	0.21375	0.21375	0.215	0.20875	0.2125
Classifier n.11	0.2125	0.2175	0.21	0.20875	0.22125
Classifier n.12	0.215	0.21375	0.215	0.2125	0.22875
Classifier n.13	0.21375	0.21625	0.2125	0.21125	0.22125
Classifier n.14	0.21375	0.215	0.20625	0.215	0.2225
Classifier n.15	0.21625	0.21125	0.21	0.21875	0.21625
Classifier n.16	0.21	0.21375	0.21125	0.215	0.22
Classifier n.17	0.2125	0.21875	0.2125	0.21625	0.23125
Classifier n.18	0.22	0.2175	0.22125	0.21	0.22125
Classifier n.19	0.22	0.21625	0.21	0.22125	0.2125
Classifier n.20	0.2175	0.21375	0.20625	0.22	0.21625
Testing - Metrics					
Recall	0.59375	0.5625	0.5781	0.6094	1.0
Precision	0.5588	0.5625	0.5286	0.5270	0.32
Accuracy	0.72	0.72	0.7	0.7	0.32
Confusion Matrix	$\begin{bmatrix}106 & 26\\ 30 & 38\end{bmatrix}$	$\begin{bmatrix}108 & 28\\ 28 & 36\end{bmatrix}$	$\begin{bmatrix}103 & 27\\ 33 & 37\end{bmatrix}$	$\begin{bmatrix}101 & 25\\ 35 & 39\end{bmatrix}$	$\begin{bmatrix}0 & 0\\ 136 & 64\end{bmatrix}$

Table 2: AdaBoost Results for Different β Values

Note: the first row and the first column of the confusion matrix correspond to the predicted/real class 0 and the others correspond to the class 1.
If $\beta = 0$ we're not weighting the samples, that is we're not improving our model and the AdaBoost implementation is pointless: we're just taking our classifier and train it on a dataset in which each data point has the same weight: the algorithm becomes equivalent to a basic majority voting ensemble.
From the confusion matrix, we can see that in the case with $\beta \neq 0$, no instance is classified as 0. This is because when we weight the predictions of the various models, we multiply the results by α, that is defined as $\beta \cdot \ln((1-\varepsilon)/\varepsilon)/2$; so, every prediction ends up to be 0, that corresponds (in the code) to the predicted class 1.
During the training, the error rate slightly improves only in the case $\beta = 0.5$, I've tried to increment the number of classifier (longer training) but I didn't obtain significant evolutions in the error rates.
By looking at the distribution of classes in the dataset, I noticed that the classes are unbalanced (Figure 3). In this situation, the AdaBoost algorithm may give more emphasis to the minority class during training, which can lead to sub-optimal performances.

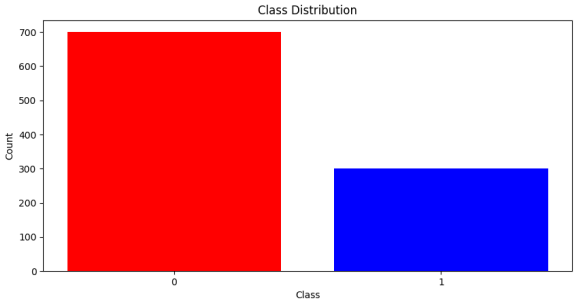

Figure 3: Class distribution of the *creditDE* dataset

4 Task 4

These are the performances of bagging applied to a SVM with a **linear kernel** on the credit dataset, using 20 classifiers:

Metric	Value
Recall	0.547
Precision	0.493
Accuracy	0.675
Confusion Matrix	$\begin{bmatrix} 100 & 29 \\ 36 & 35 \end{bmatrix}$

Table 3: Bagging applied to SVM with linear kernel

These are the performances of bagging applied to a SVM with a **RBF kernel** on the credit dataset, using 20 classifiers:

Metric	Value
Recall	1.0
Precision	0.281
Accuracy	0.285
Confusion Matrix	$\begin{bmatrix} 1 & 0 \\ 143 & 56 \end{bmatrix}$

Table 4: Bagging applied to SVM with RBF kernel

In this case, we can see that the bagging method is not efficient.

Let's try to apply the AdaBoost over this dataset, the parameter selected is $\beta = 0.5$, the classifier are 20 for the execution on the linear kernel and 15 for the RBF kernel (due to computational reasons). The results are shown in Table 5.

We can see that also with AdaBoost we have problems in the classification in the RBF kernel case. Moreover, the error rate increase during the training: SVM might be influenced by the majority class, and therefore have difficulties in correctly classifying the minority class.

Kernel	**Linear**	**RBF**
Training	**Error rate**	**Error rate**
Classifier n.1	0.22	0.1175
Classifier n.2	0.2225	0.16125
Classifier n.3	0.21125	0.19
Classifier n.4	0.2175	0.22125
Classifier n.5	0.22125	0.23625
Classifier n.6	0.21875	0.2475
Classifier n.7	0.22125	0.265
Classifier n.8	0.22	0.26875
Classifier n.9	0.22375	0.27375
Classifier n.10	0.22125	0.27875
Classifier n.11	0.21125	0.28875
Classifier n.12	0.2225	0.28875
Classifier n.13	0.21875	0.2925
Classifier n.14	0.22125	0.30125
Classifier n.15	0.21875	0.29875
Classifier n.16	0.21625	0.29875
Classifier n.17	0.215	0.30125
Classifier n.18	0.215	0.29875
Classifier n.19	0.22	0.30125
Classifier n.20	0.22125	0.29875
Testing - Metrics		
Recall	0.5312	0.0
Precision	0.6297	0.0
Accuracy	0.75	0.72
Confusion Matrix	$\begin{bmatrix} 116 & 30 \\ 20 & 34 \end{bmatrix}$	$\begin{bmatrix} 144 & 56 \\ 0 & 0 \end{bmatrix}$

Table 5: AdaBoost Results for Linear and RBF Kernels