Support Vector Machines and Ensemble Methods

Algorithmic Data Analysis – Coding Assignment 1

Giulia Ortolani

February 2, 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).

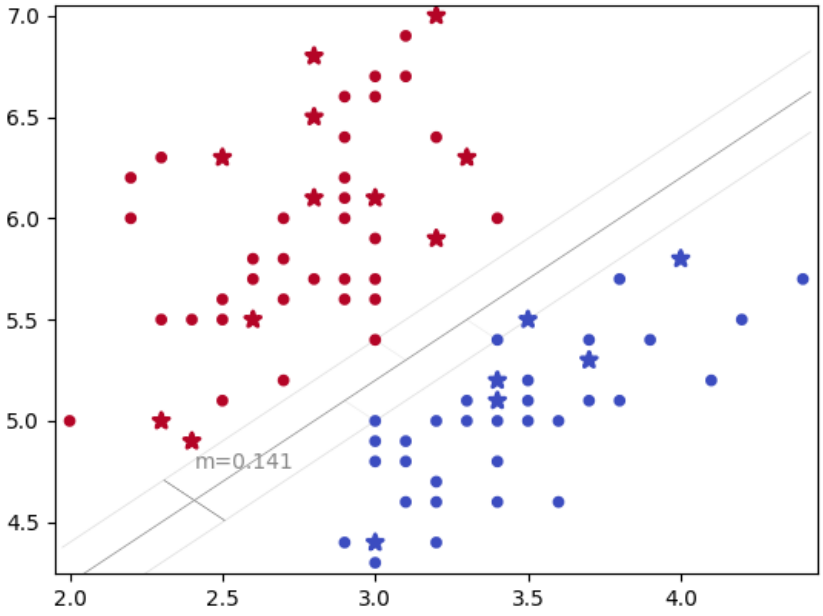

Figure 1: SVM with hard-margin – irisSV dataset

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).

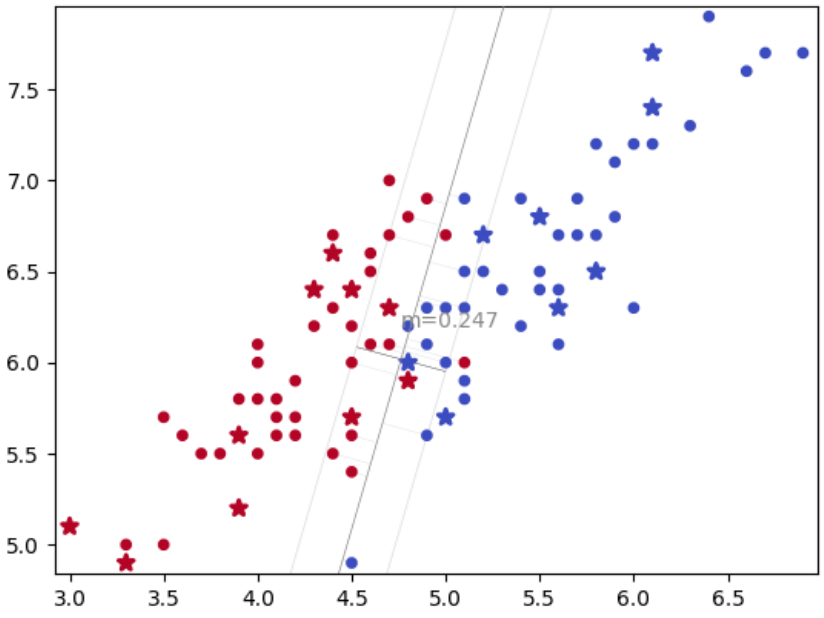

Figure 2: SVM with soft-margin – irisVV dataset

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

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

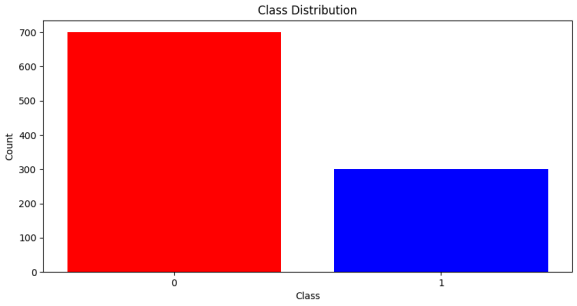

Figure 3: Class distribution of the *creditDE* dataset

during training, which can lead to sub-optimal performances. So, I tried to oversample the minority class, and I trained the AdaBoost on a dataset with final class distribution shown in Figure 4.

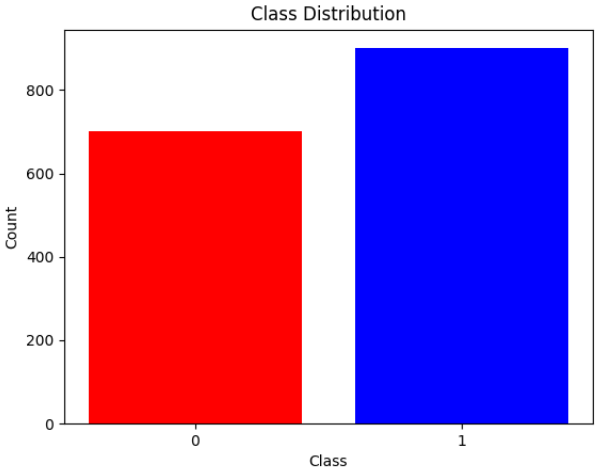

Figure 4: Dataset after oversampling

In Table 2 are shown the results of applying AdaBoost with different value for β:

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 it is equivalent to a bagging strategy.

From the confusion matrix, we can see that in the case with $\beta = 0$, 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 is slightly improving in almost all cases.

Beta	**1**	**0.8**	**0.5**	**0.2**	**0**
Training	**Error rate**	**Error rate**	**Error rate**	**Error rate**	**Error rate**
Classifier n.1	0.2484	0.2477	0.2320	0.2375	0.2656
Classifier n.2	0.2414	0.2352	0.2305	0.2313	0.2453
Classifier n.3	0.2375	0.2305	0.2281	0.2367	0.2594
Classifier n.4	0.2336	0.2305	0.2266	0.2320	0.2313
Classifier n.5	0.2383	0.2383	0.2266	0.2313	0.2453
Classifier n.6	0.2336	0.2297	0.2273	0.2391	0.2383
Classifier n.7	0.2344	0.2320	0.2289	0.2375	0.2414
Classifier n.8	0.2344	0.2328	0.2281	0.2383	0.2391
Classifier n.9	0.2336	0.2367	0.2266	0.2273	0.2352
Classifier n.10	0.2336	0.2328	0.2234	0.2313	0.2484
Classifier n.11	0.2398	0.2313	0.2328	0.2344	0.2359
Classifier n.12	0.2445	0.2320	0.2258	0.2352	0.2414
Classifier n.13	0.2383	0.2406	0.2305	0.2375	0.2359
Classifier n.14	0.2398	0.2383	0.2313	0.2375	0.2352
Classifier n.15	0.2391	0.2359	0.2320	0.2367	0.2469
Classifier n.16	0.2359	0.2352	0.2320	0.2344	0.2453
Classifier n.17	0.2375	0.2313	0.2328	0.2328	0.2313
Classifier n.18	0.2383	0.2359	0.2320	0.2336	0.2406
Classifier n.19	0.2383	0.2398	0.2328	0.2383	0.2297
Classifier n.20	0.2430	0.2375	0.2359	0.2344	0.2445
Testing - Metrics					
Recall	0.9100	0.9260	0.9470	0.9260	0.995
Precision	0.6560	0.6730	0.6490	0.6630	0.614
Accuracy	0.6660	0.6920	0.6670	0.6780	0.628
Confusion Matrix	$\begin{bmatrix} 41 & 17 \\ 90 & 172 \end{bmatrix}$	$\begin{bmatrix} 46 & 14 \\ 85 & 175 \end{bmatrix}$	$\begin{bmatrix} 34 & 10 \\ 97 & 179 \end{bmatrix}$	$\begin{bmatrix} 42 & 14 \\ 89 & 175 \end{bmatrix}$	$\begin{bmatrix} 13 & 1 \\ 118 & 188 \end{bmatrix}$

Table 2: AdaBoost Results for Different β Values

4 Task 4

Also in this case, before executing the algorithms, the dataset was balanced in the same way as Task 3.

In Table 3 are shown the performances of bagging applied to **SVM** on the credit dataset, using 15 classifiers:

In Table 4 are shown the performances of bagging applied to a **SVM with a RBF kernel** on the credit dataset, using 15 classifiers:

Let's try to apply the AdaBoost over this dataset, the parameter selected is $\beta = 0.5$, the trained classifiers in both cases are 15. The results are shown in Table 5. In this case, AdaBoost performs really well when applied to the SVM method. 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.

Metric	Value
Recall	0.788
Precision	0.730
Accuracy	0.703
Confusion Matrix	$\begin{bmatrix} 76 & 40 \\ 55 & 149 \end{bmatrix}$

Table 3: Bagging applied to SVM

Metric	Value
Recall	0.9
Precision	1.0
Accuracy	0.941
Confusion Matrix	$\begin{bmatrix} 130 & 19 \\ 0 & 171 \end{bmatrix}$

Table 4: Bagging applied to SVM with RBF kernel

	SVM	**RBF kernel**
Training	**Error rate**	**Error rate**
Classifier n.1	0.2469	0.1175
Classifier n.2	0.2414	0.1613
Classifier n.3	0.2375	0.19
Classifier n.4	0.2344	0.2213
Classifier n.5	0.2430	0.2363
Classifier n.6	0.2414	0.2475
Classifier n.7	0.2438	0.265
Classifier n.8	0.2414	0.2688
Classifier n.9	0.2375	0.2738
Classifier n.10	0.2406	0.2788
Classifier n.11	0.2383	0.2383
Classifier n.12	0.2383	0.2888
Classifier n.13	0.2406	0.2925
Classifier n.14	0.2391	0.3013
Classifier n.15	0.2445	0.2988
Testing - Metrics		
Recall	0.9	1.0
Precision	1.0	0.593
Accuracy	0.94	0.593
Confusion Matrix	$\begin{bmatrix} 130 & 19 \\ 0 & 171 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 130 & 190 \end{bmatrix}$

Table 5: AdaBoost Results for Linear and RBF Kernels