Report introML 29-11

Lorenzo Beltrame

26 October 2021

1 Task 1

The data are divided into train test (0.7/0.3) and then scaled using a MinMaxScaler.

1.1 Subtask 1

In the first subtask we were asked to implement a greedy forward feature selection with respect to the cross validated support vector machine classifier. The score function that I decided to use is the f1.

I first fixed a seed (42 :)) since I want to have a reproducible results!

The feature that are extracted are, in order of selection: 'mean radius', 'mean perimeter', 'mean concavity', 'mean compactness', 'mean area', 'mean texture', 'perimeter error', 'worst texture', 'symmetry error', 'worst radius', 'concave points error', 'radius error', 'worst area', 'smoothness error', 'mean smoothness', 'worst perimeter', 'area error', 'mean concave points', 'concavity error', 'mean fractal dimension', 'worst symmetry', 'mean symmetry', 'fractal dimension error', 'worst fractal dimension', 'worst compactness', 'worst smoothness', 'worst concavity', 'compactness error', 'worst concave points', 'texture error'.

To represent the features with respect to the increase in the f1 score[1]:

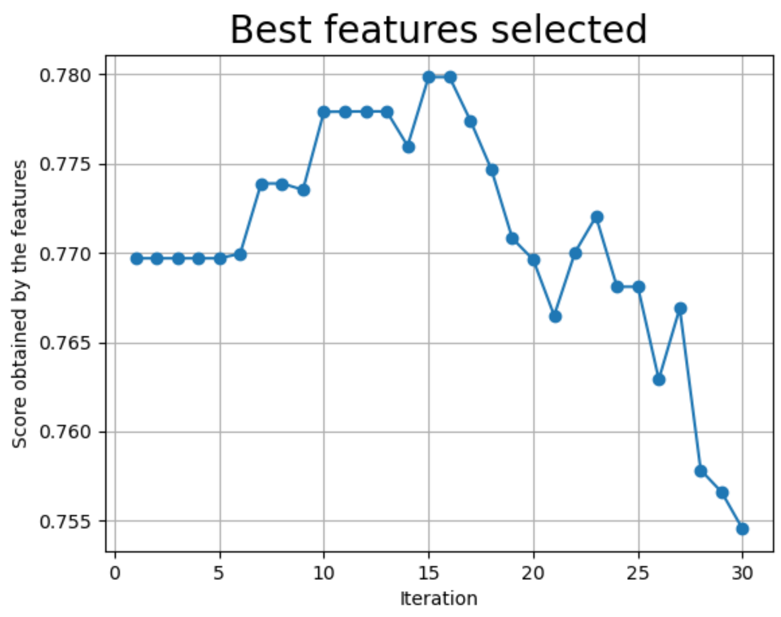

Figura 1: Graph presenting the results of the Forward feature selection. Each iteration add a feature to the subset of features we are using with respect to the CV SVC. The score chosen is the F1 score.

[1]I chose to use the f1 score as the metric since it takes into consideration the number of FP and FN, since it is really nice for binary data. Moreover in the Breast Cancer dataset, the two classes are more or less 1/3 of the total for the malign cancers and 2/3 for the benign. (212(M),357(B))

1.2 Subtask 2

In the second subtask I implement the backward feature selection with respect to the cross validated support vector machine classifier. The score function that I decided to use is the f1 score.

The feature eliminated are, in order: 'mean symmetry', 'concavity error', 'worst texture', 'mean radius', 'mean perimeter', 'mean concavity', 'mean smoothness', 'worst smoothness', 'mean area', 'smoothness error', 'mean concave points', 'mean fractal dimension', 'radius error', 'fractal dimension error', 'texture error', 'perimeter error', 'concave points error', 'worst concavity', 'mean compactness', 'worst concave points', 'worst perimeter', 'worst area', 'worst symmetry', 'worst fractal dimension', 'area error', 'symmetry error', 'compactness error', 'mean texture', 'worst radius', 'worst compactness'.

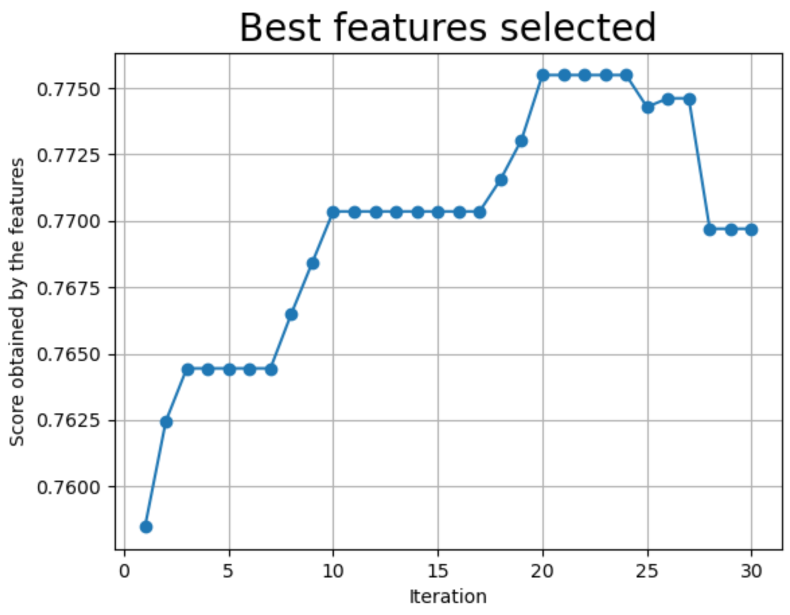

Figura 2: Graph presenting the results of the backward feature selection. Each iteration subtracts a feature from the initial subset of features. As a learning algorithm we decided to use a CV SVC. The score chosen is the F1 score.

1.3 Subtask 3

If I am supposed to use just 6 of the selected features I would use the first six obtained with the Forward FS and the last six obtained with the Backward FS. Those, with respect to their algorithms, select the most meaningful information that we can have with only 6 features.

- Forward FS: 'mean radius', 'mean perimeter', 'mean concavity', 'mean compactness', 'mean area', 'mean texture

- Backward FS: 'area error', 'symmetry error', 'compactness error', 'mean texture', 'worst radius', 'worst compactness'

The feature that appear in both the selected feature is "mean texture". The difference between the features might be explained by the fact that the backward FS starts with generally more data, therefore it might be able to recognise deeper pattern in our data. This is visible since the backward algorithm is more stable through iterations. But its main drawback is its computational cost: forward FS is generally faster.

2 Task 2

2.1 Subtask 1

In this subtask we worked with the 20 Newsgroup scikit dataset.

The data set contains vectorized test data (more than 100k features) distributed over 20 different classes.

The first task was to evaluate the accuracies of a vanilla SVC on both the test and the train set for the following kernel: linear, poly, rbf, sigmoid.

The results were the following

- Accuracies for the vanilla linear kernel: TRAIN: 0.8364857698426728 TEST: 0.5624004248539565
- Accuracies for the vanilla poly kernel: TRAIN: 0.9622591479582818 TEST: 0.49960169941582583
- Accuracies for the vanilla rbf kernel: TRAIN: 0.9314124094042779 TEST: 0.5415560276155071
- Accuracies for the vanilla sigmoid kernel: TRAIN: 0.6420364150609864 TEST: 0.47158789166224113

To select the best parameters I opted to use a scikit's grid search CV. Since we are working with different kernels and the dataset is really large I defined a grid for each type of kernel, tuning only the meaningful parameters (for further reference check get_param_grid function in Functions and scikit SVC documentation).

I chose to use a 3 fold CV since adding more folds would have slowed down the process significantly.

As a first rough part I run the grid search algorithm providing logarithmicly spaced hyper parameters. This part took more or less nine hours, but enabled me to select a better range for my hyper parameters. Subsequently, I set the various grid as visible in the get_param_grid function, in order to select hyper parameters with a better resolution. This process is definitely quicker, it takes more or less a couple of hours.

I obtained the following results:

Accuracies for the tuned linear kernel: TRAIN: 0.8364857698426728 TEST: 0.5624004248539565 With the following hyperparameters: With the following hyperparameters: 'decision_function_shape': 'ovo'

Accuracies for the tuned poly kernel: TRAIN: 0.9671203818278239 TEST: 0.5826967604885821 With the following hyperparameters: 'coef0': 1, 'degree': 2, 'gamma': 1

Accuracies for the tuned rbf kernel: TRAIN: 0.9284072830121973 TEST: 0.5419543281996814 With the following hyperparameters: 'gamma': 1

Accuracies for the tuned sigmoid kernel: TRAIN: 0.6591833127099169 TEST: 0.48154540626659587 With the following hyperparameters: 'coef0': 0, 'gamma': 1

The only result that is definitely better than the vanilla ones is the polynomial tuned kernel. The other kernels lead to results that are only slightly better than the not cross validated. This probably means that the parameter grid should have been more densly populated. The obvious downside of that is that the computational time becomes too high. A nice alternative might be to develop a custom kernel![2]

NOTE: in this part I used the function "parallel_backend" from the Joblib library in order to speed up the computations during the cross validation.

[2]To see how the CV bettered the result consider the difference between CV accuracies and vanilla accuracies.

2.2 Subtask 2

In this second part we were supposed to develop out own kernel, with the condition that it had to be yield to better result with respect to the Cross Validated kernels.

The learning problem involved the use of really large sparse matrices, representing the occurrences of words mapped into an English dictionary. The results were normalized such that all the instances had unitary norm. The idea of my kernel is to use the matrix product between the design matrix (either the train or the test) and the train design matrix. This basically led to sum the occurrences of a given word (i.e. a column in the design matrix) in all the articles we analysed (note that each instance of the design matrix is an article).

This might already be a nice feature, but we might do better by weighting each kernel with a tangent function. In particular we consider $k(X, X') =$ C $tan(X^T X')$, where C is a multiplicative hyper parameter.[3]

If we feed the result of the dot product between a row and a column into a tangent we will "value" more the words that are significantly more present in certain semantic areas (for example all the words that are connected to guns in the talk.politics.guns newsgroup).

I included the C hyper parameter to try to deal with the normalization applied to the vectorized data set. I did not work with the unvectorized vesion of the dataset because I have experience in NLP.

The results are slightly better than the cross validated ones, probably better accuracies are possible if a cross validation for SVC parameters (and also C!) is performed. In this case I manually selected C = 1.4.

Accuracies for the nice kernel: TRAIN: 0.9686229450238643 TEST: 0.5836431226765799

The improvement with this manually tuned kernel is not that relevant, remaining nevertheless slightly superior, but cross validated results might lead to significant increases in the test accuracy!

[3]Note that K respects the "kernel rules" we defined during the lectures.