TP 1 - Apprentissage Statistique Appliqué

Nokri Amale, Rahis Erwan, Vuillemot Bertrand

2020 - 2021

Contents

1 Part 1

1.1 Cross-Validation with GridSearchCV

Explain in your report what happens when we run clf.fit(X_train, Y_train)

The line `clf.fit(X\_train, Y\_train)` here uses the fit method on the object clf and takes as parameters the labels and features of the train sample. The fit method is training the model defined in the clf object. The object clf is from the class GridSearchCV which allows us to find the best hyperparameters among a fixed list we chose and perform a cross-validation. It is taking as parameters an object we §named knn of the class KNeighborsClassifier(), a dictionary named parameters containing the number of neighbors to be tested in the knn algorithm (1 to 5 here) and the cv parameter referring to the number of folds to be used in the cross-validation. Basically it will perform a 3-folds cross-validation on a kNN model with 1 to 5 neighbors on the train sample and it will allow us to keep the best model. The kNN algorithm is parametered with the default metric which is the Euclidean distance : $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. The functions are all part of the sklearn package.

What is the complexity for each of the three following cases?

Complexity can be divided into two kinds of complexity i.e: 1) time complexity, deal with how long the algorithm is executed, and 2) space complexity, deal with how much memory is used by this algorithm.

Table 1: Complexity

	kNN	Linear SVC	Log Reg
Time Training	O(n*k*d)	O(m*n)	O(n*d)
Space	O(n*d)	O(l)	O(d)

With n : size of the training sample, d : dimension of the data, k : number of neighbors, m : number of features, l : support vectors.

What is the test accuracy? What would be the accuracy of random guess?

The test accuracy is the measure of how often the points are correctly classified in the test sample. In our case the accuracy is 0.875. It means that 87.5% of the time, the points are correctly classified on the test sample. It is computed as the number of well classified individuals over the sample size. If we did a random guess we would randomly choose an output in the range 0 to 9 so the accuracy would converge towards $\frac{1}{10}$ according to the LLN.

What is LinearSVC() classifier? Which kernel are we using? What is C? (this is a tricky question, try to find the answer online)

LinearSVC means Linear Support Vector Classification, which is supervised learning methods used for classification. LinearSVC are classes capable of performing binary and multi-class classification on a dataset. This classifier is trained to separate the True labels from the False labels with a boundary compute with a kernel function. In our multi-class problem, the algorithm creates one classifier for each class and performs a "one-vs-rest" training by training the data of one class versus all the the other data as one other class.
The linear SVC is using a linear kernel function of the form : $\langle x, x' \rangle$ which implies a boundary of linear form[1].
The parameter C is the regularisation parameter. It is a parameter applied to the squared hinge loss[2] distance: $\max(0, 1 - y \cdot y')^2$ and it allows to control the importance given to good classification (accuracy) when increasing its value with respect to the margin size. When decreasing its value the accuracy is decreased but it increases the importance of a larger margin between data points and thus a simpler decision function.

[1] Appendix 3.1.1, figure 1
[2] This is true for Linear SVC. Non-linear SVC will use the l2 distance.

What is the outcome of np.logspace(-8, 8, 17, base=2)? More generally, what is the outcome of np.logspace(-a, b, k, base=m)?

The outcome of np.logspace(-8, 8, 17, base=2) is a logarithmic space going from 2^{-8} to 2^{8} with 17 numbers equally spaced on log scale. The logspace function from the numpy package will return k numbers going from m^{-a} to m^{b} spaced on a log scale with a log base m.

What is the meaning of the warnings? What is the parameter responsible for its appearance?

The warning tells us that the algorithm did not converge, it did not reach the stop criterion[3] before the number of maximum iterations. The parameter responsible for its appearance is the max_iter parameter. Its value is not large enough for the algorithm to converge. The data variance is maybe too large for the algorithm to efficiently perform the SVM.

What did we change with respect to the previous run of LinearSVC()?

We added a pipeline which is a method of the ScikitLearn package that allows to streamline the data pre-processing. In the pipeline we added a `MaxAbsScaler()` method to scale the absolute data between 0 and 1 and thus reduce the variance of the data. We notice that the algorithm is not showing a convergence warning.

Explain what happens if we execute :

```
pipe.fit(X_train, y_train)
pipe.predict(X_test, y_test)
```

The first will execute the pipeline defined with a `MaxAbsScaler` preprocessing on the features and fit a SVM but this time with no C parameter defined which will be 1.0 by default. The second line is returning an error. Indeed, the predict method is returning an array of predictions made with the pipe object and should only take one argument, that is an array of features (X). Here we put two arguments as parameters and it is the reason why it is returning an error.

What is the difference between `StandardScaler()` and `MaxAbsScaler()`? What are other scaling options available in sklearn?

StandardScaler will standardize the data : $\frac{x-m}{\sigma}$ with m the mean and σ the standard deviation of data. It differs from MaxAbsScaler because in this case we map the absolute value of data in a [0,1] range.

Two of the other scaling options available in sklearn are :
- `MinMaxScaler` which transforms features by scaling each feature to a given range [min, max].
- `RobustScaler` which is used if the data contains many outliers. It basically removes the data median and then scales the data according to the quantile range which is the range between the 1st quartile(25%) and the 3rd quartile (75%).

Using the previous code as an example achieve test accuracy ≥ 0.9 . You can use any method from sklearn package. Give a mathematical description of the selected method. Explain the range of considered hyper-parameters.

We tried the Random Forest algorithm which is a method creating a fixed number of random trees (CART algorithm). The randomness in this algorithm comes with the selection of features used to create the trees. Each tree is created with a fixed number of features but these features are randomly drawn from the whole range of available features.

In our case, the dataset has 784 features and the algorithm choses $\sqrt{784} = 28$ features for each tree. This function also uses the bagging method for the elements of the sample. It means that for each tree it takes a random sample of the same size as the initial sample. In this case we fit a train sample of

[3]The stop criterion here is the tol parameter

size 2000 so the bootstrap bags will have 2000 random elements (they can appear multiple times). In each tree the method is to successively split the features into 2 groups. The choice of the feature and threshold for the split is made by minimising a criterion : the gini coefficient or the entropy. In our case we put both hyper-parameters for the Grid Search to find the best one.

We used the method `RandomForestClassifier()` from the skLearn package in the pipeline along with a `StandardScaler` preprocessing on features to normalise the data and reduce the variance so we avoid divergence of the algorithm. The number of trees to generate and the split quality criterion are the two hyper-parameters we chose to exploit. The default number of trees is 100 so we tried with 50 and 150. We used the accuracy scoring for the grid search and a 3 folds cross-validation. This configuration resulted in an accuracy > 0.9.

1.2 Visualising errors

The error in the chunk of code was because the `predict_proba` method returns an array of probabilities within an array. We must then pick the first element of the array (index 0) to retrieve the probabilities array[4]. The output of this code chunk is a figure with the 4 first misclassified items by the algorithm along with the probability of each class.

1.3 Changing the loss function

What is balanced_accuracy_score? Write its mathematical mathematical description.

The balanced accuracy in binary and multi-class classification problems is used to deal with imbalanced datasets. For a binary classification is defined as the arithmetic mean of the sensitivity (also called recall or true positive rate) and the specificity (also called true negative rate). As a consequence, for a multi-class classification, it represents the average recall per class. The recall score for one class i :

$$recall_i = \frac{tp_i}{tp_i + fp_i}$$

with tp_i: true positive (well classified items in the class i) and fn_i: false negative (items which should be classified in class i but were not).
Instead of calculating the regular score which is $\frac{tp+tn}{sampleSize}$, the balanced score is, for K classes, :

$$\frac{\sum_{i=1}^{K} recall_i}{K}$$

If the number in each category of prediction is the same, regular score = balanced score. Otherwise, the good predictions of an over represented class will not inflate the balanced score unlike the regular one.

What is the confusion matrix? What are the conclusions that we can draw from the `confusion_matrix(y_test, clf4.predict(X_test))`?

The general idea is to count the number of instances of class A that are classified as class B by computing a matrix giving information about true class and predicted class. For example, to know the number of times the classifier confused images of 5s with 2s, you would look in the 5th row and 2nd column of the confusion matrix. The row is the actual class and the column is the predicted class given by the algorithm.

As we can see in our case[5], 8s are often confused with 5s (3/17=18% of the time when the actual class is 8) and 3s are also confused with 5s 13% of the time (3/23). Also, 5s are detected only 57% (8/14) of the time. 0s and 9s seem well detected with respectively 100% (22/22) and 92% (24/26) recall/true positive rate.

Regarding the scores, the balanced is slightly inferior to the regular one (83% vs 84%) due to the underrepresentation of the worst predicted class (ie 5s).Because there are several classes, it could be

[4]See Jupyter Notebook File
[5]Appendix 3.1.2, figure 4

interesting to transform the confusion matrix into a heat map. On the heat map we can check that the algorithm is good at predicting classes since the brighter cells are on the main diagonal. Even though, 5s are darker than other classes explained by the under-representation of the class and a few errors. 1s are well predicted given its bright square on the main diagonal but it can be partly explained by the over-representation of 1s in the dataset.

2 Part 2

2.1 Loss Function

Our custom loss function is an accuracy score with a penalty on inter-class errors. If the predicted value is different than the true value then the error count will increase +1. We decided to separate the data into two classes :

Label	0	1	2	3	4	5	6	7	8	9
Class	0	0	0	0	0	1	1	1	1	1

The counts for each class or label i are as follows :

		Predicted	
		True	False
Actual	True	tp_i	fn_i
	False	fp_i	tn_i

We define the Recall score for one class or label i as :

$$Recall_i = \frac{tp_i}{tp_i + fn_i}$$

which is the well classified individuals on the number of individuals actually in this class. We compute a balanced accuracy on the labels (0:9) and one on the classes (0-1). The formula we use is :

$$Balanced\_Accuracy = \frac{\sum_{i=1}^{K} Recall_i}{K}$$

We join both metrics by multiplying them to have a penalised balanced accuracy of the labels :

$$Score = Balanced\_Accuracy_{Label} * Balanced\_Accuracy_{Class}$$

By this mean, our score will decrease when errors are made between two classes. For example if two algorithms show the same number of misclassified labels, it will allow to figure out which one had the most class errors and then choose the other one.

2.2 Pipeline

To scales the features, we chose to use a MaxAbsScaler on the data because we want to minimise their variance by constraining them to be between 0 and 1. In fact, we have 784 features which represent a color code that goes from 0 to 255 which can be easily scaled to a [0,1] interval without any loss of information. We train the pipe with a random forest and a SVC model. It resulted in a better score for the SVC model which is the one we are detailing in the next section. The Jupyter Notebook file contains the entire code with our trials.

2.3 Algorithm

The algorithm chosen is the Support Vector Classifier. The SVC for our multi-class problem consists in the creation of as many classifiers as there are classes, here 10 classes so 10 classifiers. Each classifier is training a class by the "one-vs-rest" method by considering one class only and the rest of the data as another class so the problem is transformed as several binary problems.

The hyper-parameters are chosen via the Grid-Search and Cross-Validation object. The hyper-parameters considered are:
⊙ Kernel : it is the Kernel function used for the decision function in the algorithm; We consider the rbf kernel : $\exp(-\gamma\|x-x'\|^2)$ and the linear kernel : $\langle x, x'\rangle$.
⊙ C : it is the regularisation parameter. It is multiplied by the squared l2 penalty $\|x-x'\|^2$ and allows to control the importance given to good classification (accuracy) by increasing its value. When decreasing its value the accuracy is decreased but it increases the importance of the larger margin between data points and thus a simpler decision function. We consider a log space as in part 1 but with 5 points.

The support vector machine separated our data with a boundary of a given form. Mathematically, it performs an optimisation with the features $x_i \in \mathbb{R}^p$, labels $y \in \{-1, 1\}^n$ and predictions : $\text{sign}(w^T\phi(x)+b)$ as follows:

$$\begin{aligned} & \min_{w,b,\zeta} \frac{1}{2}w^T w + C\sum_{i=1}^{n}\zeta_i \\ \text{s/t} \quad & y_i(w^T\phi(x_i)+b) \geq 1-\zeta_i, \\ & \zeta_i \geq 0, i = 1, ..., n \end{aligned}$$

This primal problem consists in maximising the margins by minimising $w^T w = \|w\|^2$ and penalising points that are misclassified or within the margin. We allow a distance from he correct boundary by adding the term $C\sum_{i=1}^{n}\zeta_i$. In this term, C is the regularisation parameter as explained earlier and ζ_i is the distance from the boundary. The SVC function of the sklearn package uses the l2 distance : $\|x-x'\|^2$.
The dual problem is as follows :

$$\begin{aligned} & \min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{s/t} \quad & y^T\alpha = 0 \\ & 0 \leq \alpha_i \leq C, i = 1, ..., n \end{aligned}$$

where $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ and $K(x_i, x_j)$ is the kernel function. e is a vector of ones and α are the parameters with C as upper bound. Our kernel is the gaussian radial basis function. It allows computing the output label as : $sign(\sum_{i\in SV} y_i\alpha_i K(x_i, x) + b)$. The rbf kernel function is :

$$\exp(-\gamma||x-x'||^2)$$

It is a bell-shaped function varying from 0 (very far away from the landmark) to 1 (at the landmark). The idea is to transform the dataset and by doing so, make it linearly separable through non-linear methods[6] . The metrics of the distance is usually the euclidean.

However, using the Gaussian RBF can be computationally expensive especially for large datasets given it transforms the training dataset with m instances and n features into a m instances and m features dataset. γ and C both play the role of regularisation hyper-parameter. Increasing γ makes the bell-shape narrower so it reduces the instance's range of influence which makes the decision boundary more irregular, going back and forth around individual instances. If the model is overfitting, trying to diminish the value of γ could solve the problem. In our case we chose to not consider the gamma as a hyper-parameter in the grid-search. Thus, the value used is $\gamma = \frac{1}{n\_features \times \mathbb{V}[X]}$.

[6] Appendix 3.2.1, figure 3

3 Appendix

3.1 Part 1

3.1.1 SVM illustration

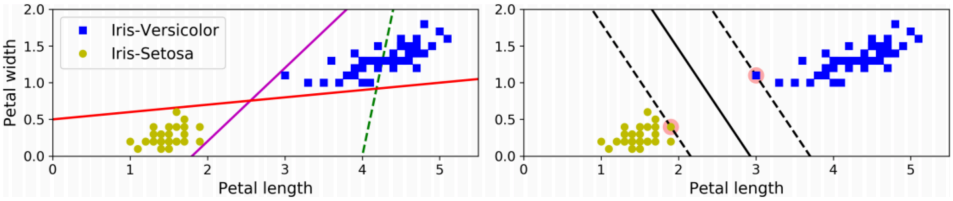

Figure 1: SVM illustration

On this figure we see two plots representing one same problem of separation. The one on the left is drawing several linear separations possible. The red and purple ones would correctly separate the data. But the purpose of the SVM is to maximise the margins around the boundary. That is why the optimised boundary is the one on the right as it maximises the distance between the closest data point and the linear boundary.

3.1.2 Changing the loss function : confusion matrix / heatmap

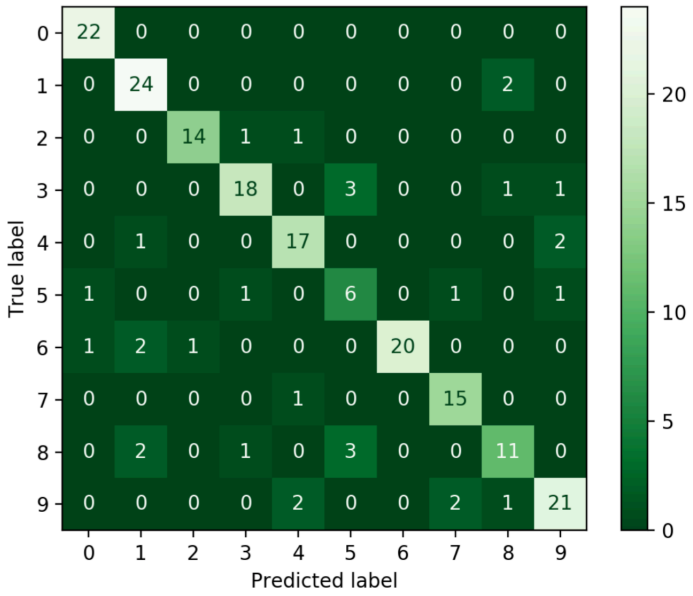

Figure 2: Confusion matrix for the SVM classifier

3.2 Part 2

3.2.1 Algorithm

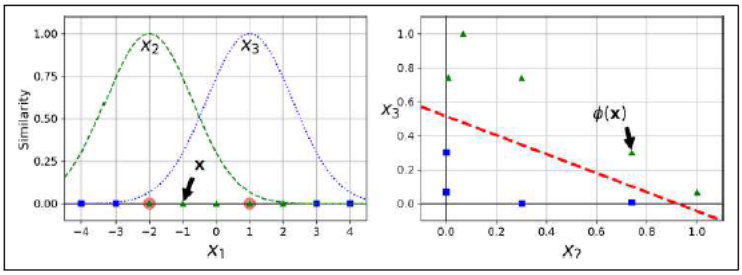

Figure 3: Similarity features using the Gaussian RBF

3.2.2 Final confusion matrix

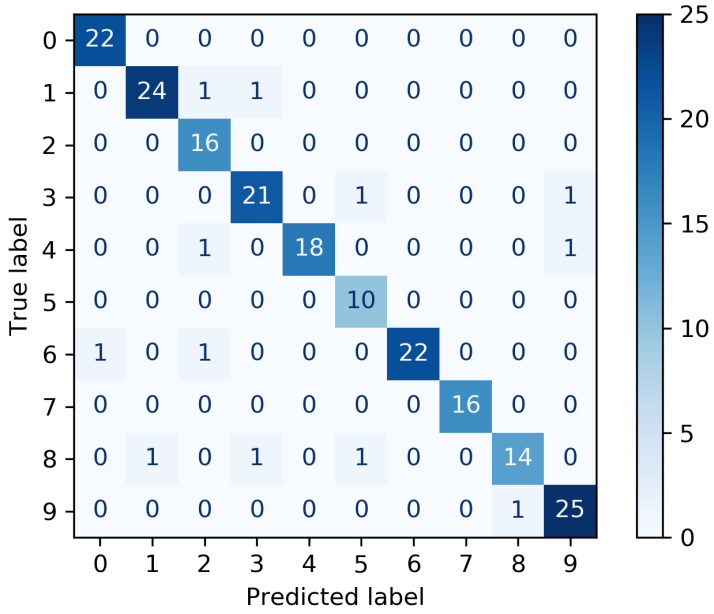

Figure 4: Confusion matrix part 2

References

[1] scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation.

[2] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., Sebastopol, CA, 2019. OCLC: 1135343456.

[3] The Kernel Trip. Computational complexity of machine learning algorithms, 2018.