TP 1 - Apprentissage Statistique Appliqué

Nokri Amale, Rahis Erwan, Vuillemot Bertrand

2020 - 2021

Contents

1 Part 1

1.1 Cross-Validation with GridSearchCV

Question: Explain in your report what happens when we run clf.fit(X_train, Y_train). What is the complexity for each of the three following cases?

The line `clf.fit(X\_train, Y\_train)` here uses the fit method on the object clf and taking the train sample. We give the features X and the outputs Y. The object clf is from the class GridSearchCV which allows us to find the best hyperparameters among a list we chose. It is taking as parameter an object 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.

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 it's 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.

Question: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 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.

Question: 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 tries to find a line that separates the True labels from the False labels.. We are using a linear kernel. The parameter C represents the regularisation weights, ie the penalty applied on the loss function. The loss function used here is the Squared Hinge Loss : $L(y, \hat{y}) = \sum_{i=0}^{N}(\max(0, 1 - y_i \cdot \hat{y}_i)^2$)

Question : 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.

Question : 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. The parameter responsible for its appearance is the max_iter parameter. Its value is not sufficient for the algorithm to converge. The data variance is maybe too large for the algorithm to efficiently perform the SVM.

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

We are running the svc function which is by default a RBF SVC and not a linear SVC. RBF means radial basis function. We added a parameter `MaxAbsScaler()` to scale the absolute data between 0 and 1 and thus reduce the variance of the data.

maybe complete definition of RBF SVC

Question : Explain what happens if we execute

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

Those lines will execute the pipeline defined with a `MaxAbsScaler` preprocessing on the features and fit a SVM but with no C parameter defined which will be 1.0 by default.

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

StandardScaler will normalise 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.
The other scaling option available in sklearn are :
- `MinMaxScaler` which transform features by scaling each feature to a given range [min, max].
- `RobustScaler`, this scale is used if your data contains many outliers.

Add other options for scaling

Question : 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 hyperparameters.

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 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 cross-validation. This configuration resulted in accuracy > 0.9[1]. We launched the fit 10 times to make sure the results are stable (as it uses random values).

definition of random forest mathematically

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[2].

1.3 Changing the loss function

Question: 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. It 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, it represents the average accuracy per class.

$$recall = \frac{tp}{tp + fp}$$

with tp: true positive and fn: false negative

$$specificity = \frac{tn}{tn + fn}$$

Instead of calculating the regular score which is $\frac{tp+tn}{sampleSize}$, the balanced score is

$$\frac{recall + specificity}{2}$$

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.

Question: 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 times instances of class A are classified as class B. For example, to know the number of times the classifier confused images of 5s

[1] See appendix 3.2
[2] See appendix 3.3, line 11

with 2s, you would look in the 5th row and 2nd column of the confusion matrix. In row the actual class and in columns the predicted class given by algorithm.

As we can see in our case[3], 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 class, it could be interesting to transform the confusion matrix into a heat map.

On the heat map we can check that the algorithm is pretty good at predicting classes since most images are on the main diagonal. Even though, 5s are darker than other classes explained by the underrepresentation of the class and the lower number of good predictions. 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 and 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 weighted accuracy on the labels (0:9) and one on the classes (0-1). The formula we use is :

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

We join both metrics by multiplying them to have a penalized weighted accuracy of the labels :

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

[3]Appendix 3.4, figure 5

2.2 Pipeline

To scales the features, we chose to use a MaxAbsScaler on the data because we want to minimize their variance knowing 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 trains a linear SVM model (using the SVC).

2.3 Algorithm

The Gaussian Radial Basis Function, or Gaussian RBF uses a similarity function that measures how much each instance resembles a particular landmark. It will create a landmark at each instance of the dataset and calculate the similarity function :

$$\varphi_\gamma(x, l) = exp(-\gamma||x - l||^2) \tag{1}$$

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 (Figure 4). 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 regularization hyperparameter. Increasing γ make the bell-shap narrower so it reduces the instance's range of influence which makes the decision boundary ends up being more irregular, wiggling around individual instances. If the model is overfitting, trying to diminish the value of γ could solve the problem.

3 Appendix

3.1 Part 1 : SVM illustration

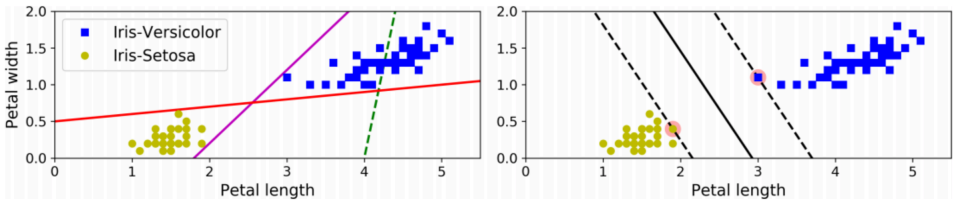

Figure 1: SVM illustration

3.2 Target accuracy : code

```
from sklearn.ensemble import RandomForestClassifier
pipe_test = Pipeline([('scaler', StandardScaler()), ('rf',
    RandomForestClassifier())])
parameters_test = {'rf__n_estimators':[100,150],
                   'rf__criterion': ['gini','entropy']}
scoring_test = {'accuracy': make_scorer(accuracy_score)}
clf_test = GridSearchCV(pipe_test, parameters_test, cv=3,scoring =
    scoring_test, refit='accuracy')
clf_test.fit(X_train, y_train)
print('Returned hyperparameter: {}'.format(clf_test.best_params_))
print('Best classification accuracy in train is: {}'.format(clf_test.
    best_score_))
print('Classification accuracy on test is: {}'.format(clf_test.score(
    X_test, y_test)))
```

3.3 Visualizing Errors : code chunk

```
axes = plt.subplots(2, 4)[1]  # creates a grid of 10 plots

# More details about zip() function here https://docs.python.org/3.3/
    library/functions.html#zip
y_pred = clf4.predict(X_test)
j = 0 # Index which iterates over plots
for true_label, pred_label, image in list(zip(y_test, y_pred, X_test)):
    if j == 4: # We only want to look at 4 first mistakes
        break
    if true_label != pred_label:
        # Plotting predicted probabilities
        axes[1, j].bar(np.arange(10), clf4.predict_proba(image.reshape(1,
    -1))[0])
        axes[1, j].set_xticks(np.arange(10))
        axes[1, j].set_yticks([])

        # Plotting the image
        axes[0, j].imshow(image.reshape((28, 28)), cmap=plt.cm.gray_r,
    interpolation='nearest')
        axes[0, j].set_xticks([])
        axes[0, j].set_yticks([])
        axes[0, j].set_title('Predicted {}'.format(pred_label)+'/True {}'.
    format(true_label),fontsize=8)
        j += 1
```

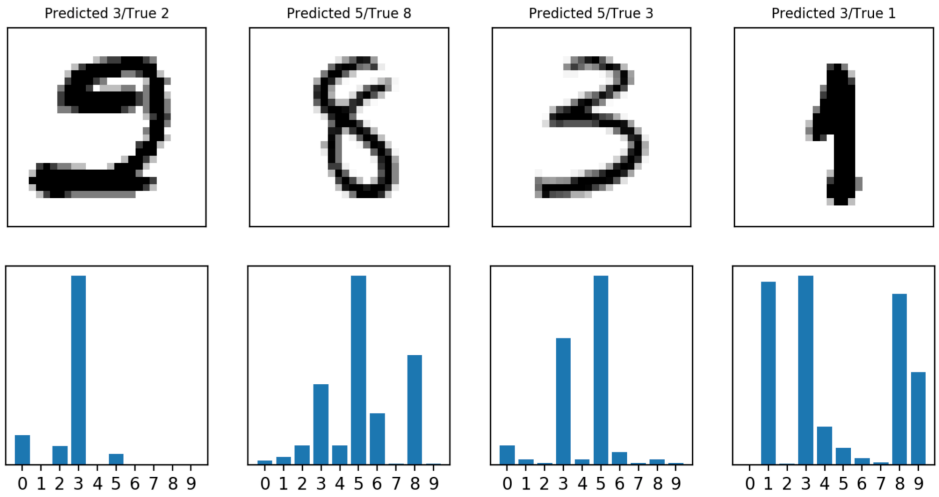

Figure 2: Probabilities of each outcome for the logistic regression

3.4 Changing the loss function : confusion matrix / heatmap

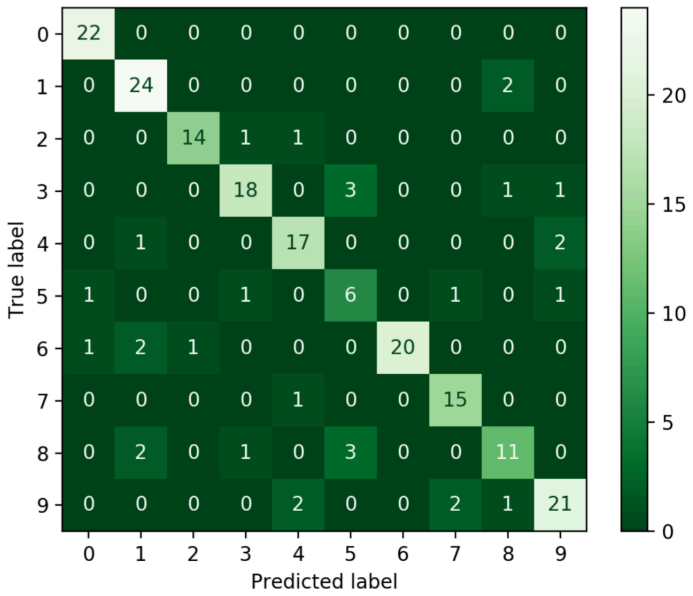

Figure 3: Confusion matrix for the SVM classifier

3.5 Part 2 : code

```
def custom_MNISTscorer(y_true, y_predict):
    #Classe 1 : les chiffres 0:4, Classe 0 : les chiffres 5:9
    y_predict_class = (np.array(y_predict).astype(int)<5).astype(int)
    y_true_class = (np.array(y_true).astype(int)<5).astype(int)

    #Compute accuracy for class 0 - 1 and labels 0:9
```

```
    Class_weightedaccuracy = sklearn.metrics.balanced_accuracy_score(
    y_true_class, y_predict_class)
    Label_weightedaccuracy = sklearn.metrics.balanced_accuracy_score(
    y_true, y_predict)
    return Label_weightedaccuracy * Class_weightedaccuracy
```

3.6 Part 2 : algorithm

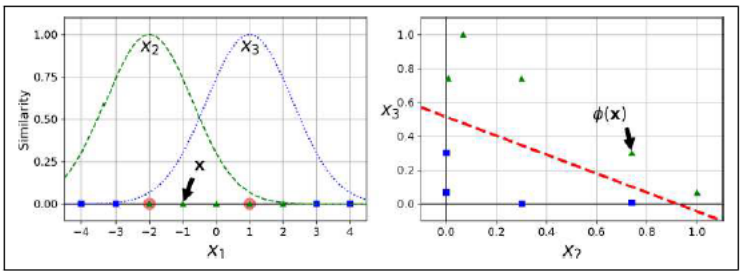

Figure 4: Similarity features using the Gaussian RBF

3.7 Part 2 : final confusion matrix

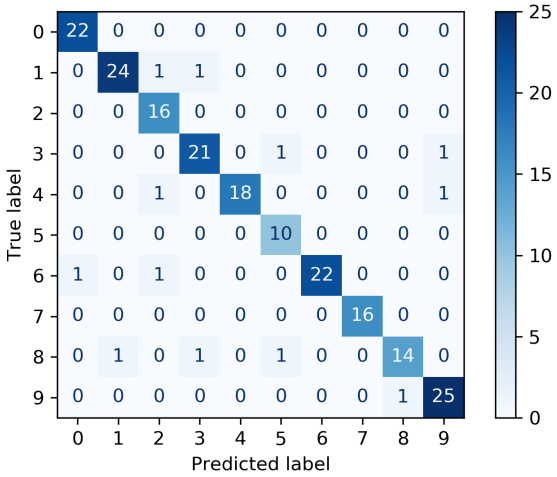

Figure 5: Confusion matrix part 2

References

[1] Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names.

[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] Elliot Tyler. How to classify MNIST digits with different neural network architectures | by Tyler Elliot Bettilyon | Teb's Lab | Medium.