Some most used classification metrics are:

1. **Confusion Matrix**: when we need to know how much samples we got right or wrong for ***each class***. The values that were correct and correctly predicted are called *true positives*, the ones that were predicted as positives but weren't positives are called *false positives*. The same nomenclature of *true negatives* and *false negatives* is used for negative values;
2. **Precision**: when our aim is to understand what correct prediction values were considered correct by our classifier. Precision will divide those true positive values by the samples that were predicted as positives;

$$precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

3. **Recall**: commonly calculated along with precision to understand how many of the true positives were identified by our classifier. The recall is calculated by dividing the true positives by anything that should have been predicted as positive.

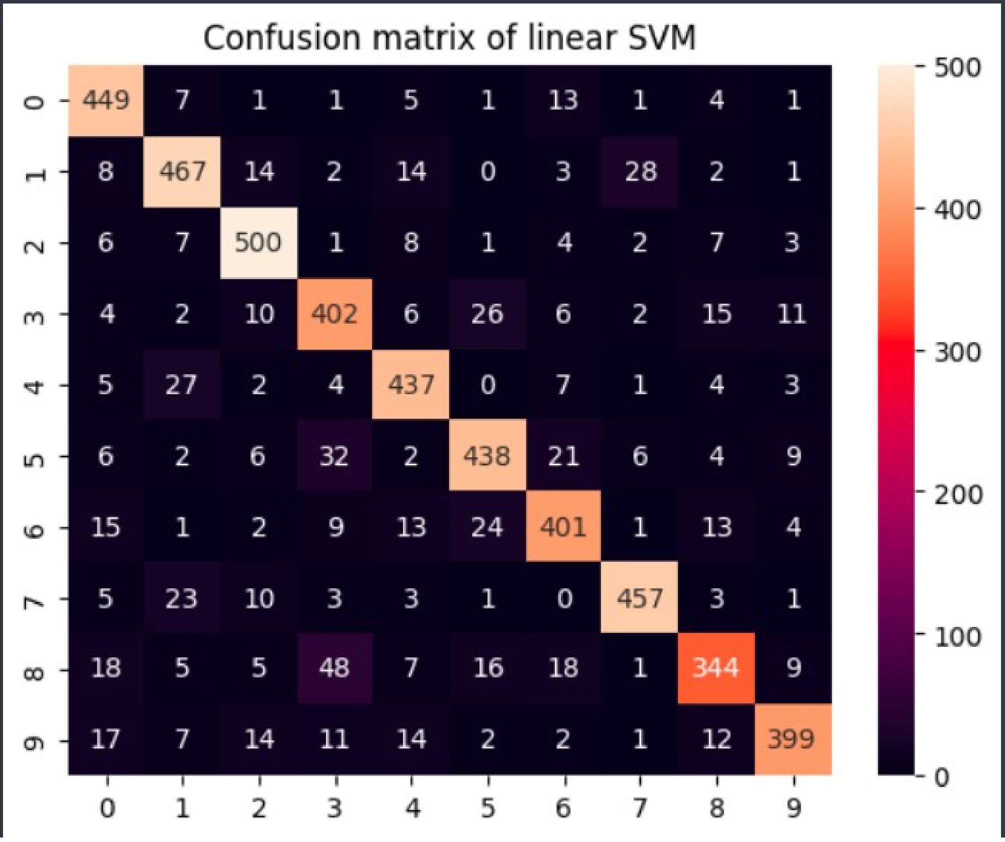
$$recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

4. **F1 score**: is the balanced or *harmonic mean* of precision and recall. The lowest value is 0 and the highest is 1. When F1 score is equal to 1, it means all classes were correctly predicted **-** this is a very hard score to obtain with real data (exceptions almost always exist).

$$\text{f1-score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

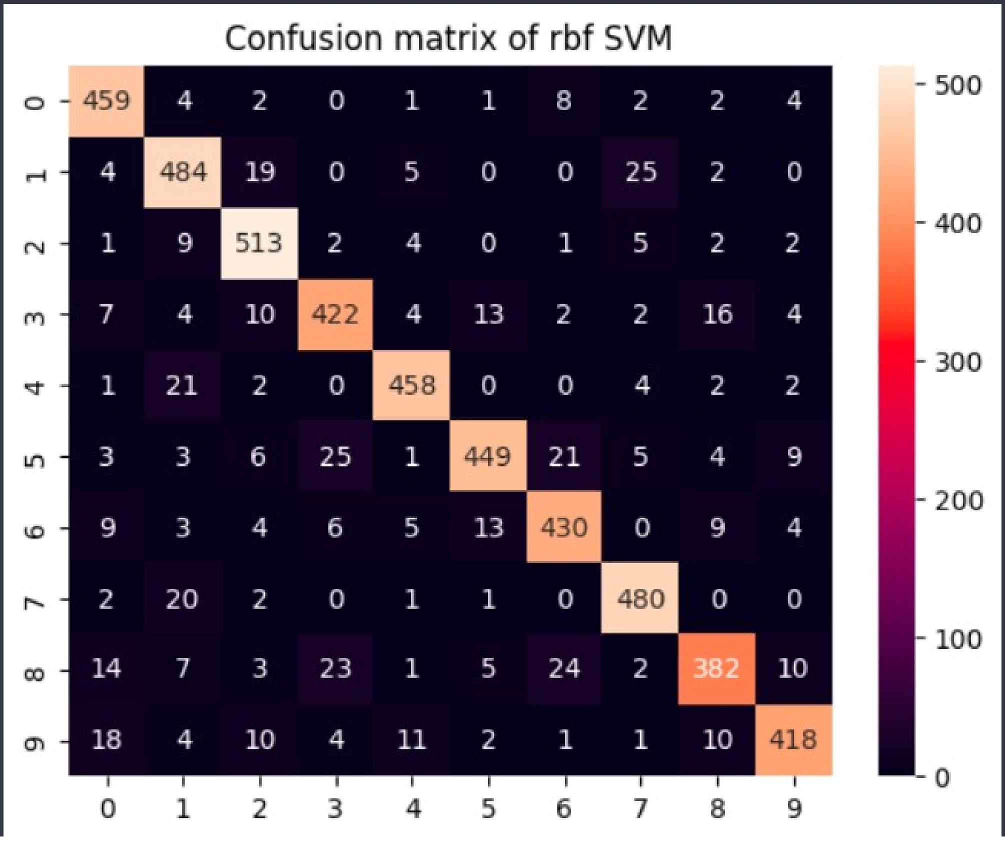
In the classification report, linear SVM has an accuracy of 0.86; RBF 0.90 and polynomial 0.87.

```
    accuracy                           0.86      5000
   macro avg       0.86      0.86      0.86      5000
weighted avg       0.86      0.86      0.86      5000
```



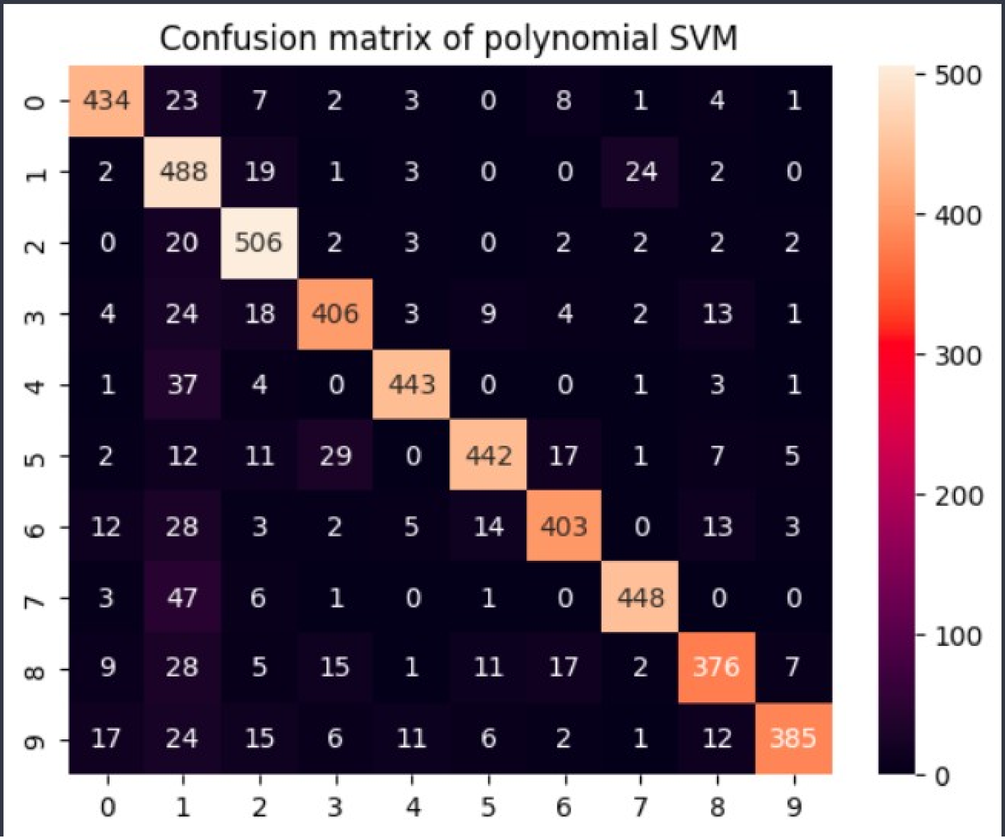
	0	1	2	3	4	5	6	7	8	9
0	449	7	1	1	5	1	13	1	4	1
1	8	467	14	2	14	0	3	28	2	1
2	6	7	500	1	8	1	4	2	7	3
3	4	2	10	402	6	26	6	2	15	11
4	5	27	2	4	437	0	7	1	4	3
5	6	2	6	32	2	438	21	6	4	9
6	15	1	2	9	13	24	401	1	13	4
7	5	23	10	3	3	1	0	457	3	1
8	18	5	5	48	7	16	18	1	344	9
9	17	7	14	11	14	2	2	1	12	399

```
    accuracy                           0.90      5000
   macro avg       0.90      0.90      0.90      5000
weighted avg       0.90      0.90      0.90      5000
```



	0	1	2	3	4	5	6	7	8	9
0	459	4	2	0	1	1	8	2	2	4
1	4	484	19	0	5	0	0	25	2	0
2	1	9	513	2	4	0	1	5	2	2
3	7	4	10	422	4	13	2	2	16	4
4	1	21	2	0	458	0	0	4	2	2
5	3	3	6	25	1	449	21	5	4	9
6	9	3	4	6	5	13	430	0	9	4
7	2	20	2	0	1	1	0	480	0	0
8	14	7	3	23	1	5	24	2	382	10
9	18	4	10	4	11	2	1	1	10	418

```
     accuracy                           0.87      5000
    macro avg       0.88      0.86      0.87      5000
 weighted avg       0.88      0.87      0.87      5000
```



	0	1	2	3	4	5	6	7	8	9
0	434	23	7	2	3	0	8	1	4	1
1	2	488	19	1	3	0	0	24	2	0
2	0	20	506	2	3	0	2	2	2	2
3	4	24	18	406	3	9	4	2	13	1
4	1	37	4	0	443	0	0	1	3	1
5	2	12	11	29	0	442	17	1	7	5
6	12	28	3	2	5	14	403	0	13	3
7	3	47	6	1	0	1	0	448	0	0
8	9	28	5	15	1	11	17	2	376	7
9	17	24	15	6	11	6	2	1	12	385

A quick test to find out if an overfit is happening is with train data. If the model has somewhat memorized the train data, the metrics will be very close to 1 or 100%.

To revert the overfit, we can add more train observations, use a method of training with different parts of the dataset, such as **cross validation**, and also change the default **hyperparameters.**

SVM Hyperparameters

To see all model parameters that have already been set by Scikit-learn and its default values, we can use the get_params() method. there are a total of 15 hyperparameters already being set.

If we can draw a line to separate our classes, then a **linear kernel** will be a good option, if we need a curve, then a **polynomial** kernel might be the best choice, if our data has circular shapes, then a **Radial Basis Function** or **RBF** kernel will suit the data better, if there are values above and below a threshold, a **sigmoid** kernel might separate the classes better.

So far based on accuracy score, we can guess that RBF is the best kernel for our dataset.

The C Hyperparameter

The C parameter is inversely proportional to the margin size, this means that the **larger** the value of C, the **smaller** the margin, and, conversely, the **smaller** the value of C, the **larger** the margin.

Our RBF kernel SVM has used a C of 1.0, which is a **large** value and gives a **smaller margin**.

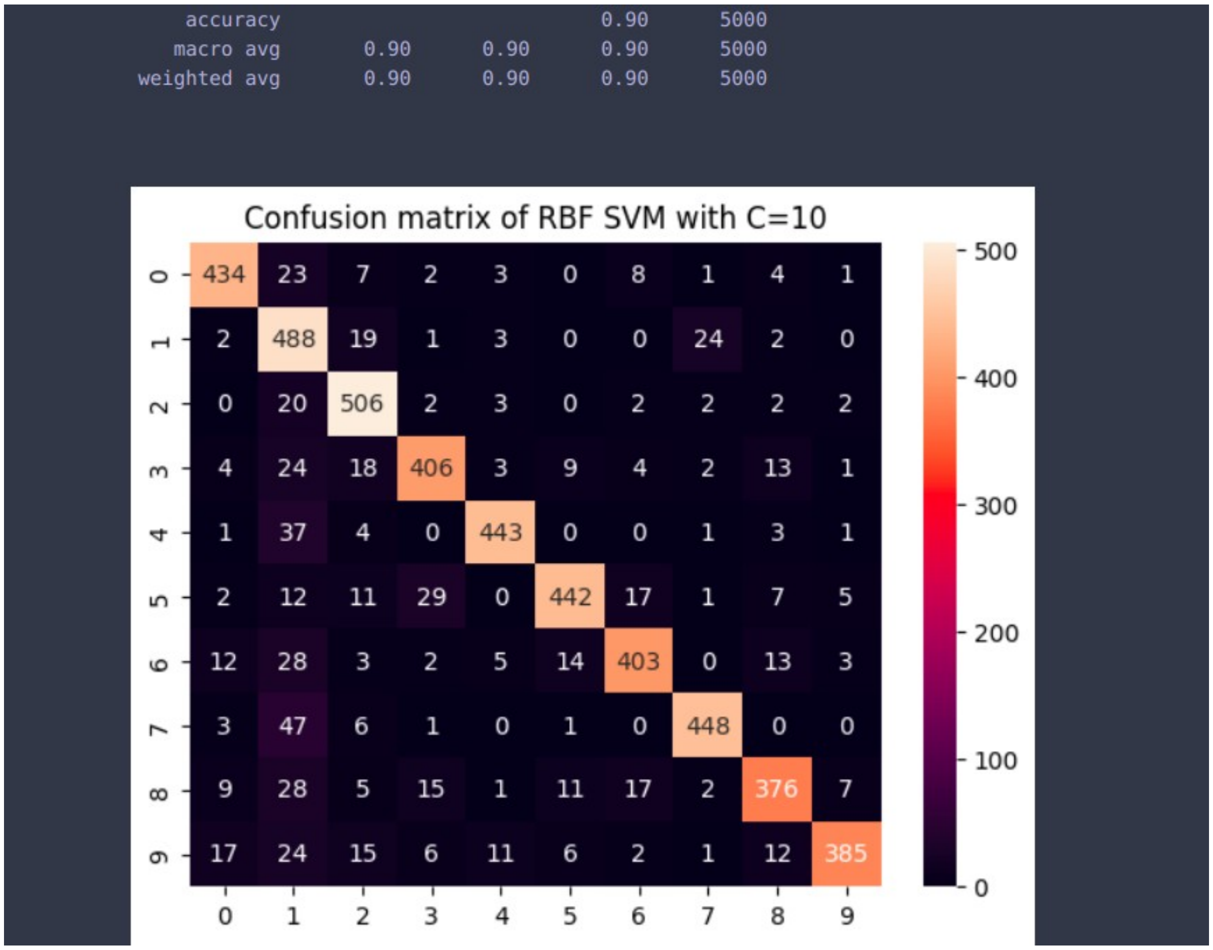
By using a smaller C and obtaining a larger margin, the classifier has become more flexible and with more classification mistakes.

The accuracy with RBF kernel and C = 10 is 0.90

The accuracy with RBF kernel and C = 1 is 0.89

The accuracy with RBF kernel and C = 0.1 is 0.72

The accuracy with RBF kernel and C = 0.01 is 0.17



```
    accuracy                           0.90      5000
   macro avg       0.90      0.90      0.90      5000
weighted avg       0.90      0.90      0.90      5000
```

Confusion matrix of RBF SVM with C=10

	0	1	2	3	4	5	6	7	8	9
0	434	23	7	2	3	0	8	1	4	1
1	2	488	19	1	3	0	0	24	2	0
2	0	20	506	2	3	0	2	2	2	2
3	4	24	18	406	3	9	4	2	13	1
4	1	37	4	0	443	0	0	1	3	1
5	2	12	11	29	0	442	17	1	7	5
6	12	28	3	2	5	14	403	0	13	3
7	3	47	6	1	0	1	0	448	0	0
8	9	28	5	15	1	11	17	2	376	7
9	17	24	15	6	11	6	2	1	12	385

The best SVM would have RBF kernel and C = 10.

Accuracy of test data is 0.90 and train data is 1.0.

The high accuracy for train data does not mean there was an overfitting; Since the model works good on test data. It means the database was easy to classify.