Python for Finance and Algorithmic Trading

Machine Learning, Deep Learning, Time Series Analysis, Risk and Portfolio Management for Metatrader™ 5 Live Trading

POWERED BY

Quantreo

Lucas Inglese

Book Link : https://www.amazon.com/gp/product/B09HG18CYL

EXTRACT Chapter 9: Support vector machine (SVM)

In this chapter, we will see a very used algorithm in finance, the support vector machine. It is advantageous in finance because it does not need many data to train. We will see how to standardize the data for a machine learning algorithm and why. Then, we will learn how to use a support vector machine regressor (SVR) and a support vector machine classifier (SVC).

9.1. Preparation of data

In this section, we are going to prepare the data for our SVM[1]. We will recap what we already know about data preparation, and we will go deeper into the concept explaining the standardization of the data.

9.1.1. Features engineering

In this section, we are going to do a quick recap of the data preparation. Indeed, we will do the same as in the previous chapter. So, we need to define sets of training and sets of tests. Furthermore, we will improve the features of our model. To do it, we will add some technical indicators. We do not explain the technical indicators we will use because they are not relevant to the book. If you want to find some documentation about this, you can find it on internet and in books.

[1] **Additional lecture** : Support-vector machine, Wikipedia

Book Link : https://www.amazon.com/gp/product/B09HG18CYL

Features engineering helps the algorithm to find the best pattern in the data. For example, the stock price is not reasonable because if you have an asset with a price between 50$ and 5000$, the model's parameters cannot be fit correctly, instead of with the percentage variation which turns all assets into the same range. It is very helpful for the algorithm.

To create technical indicators, there are two ways. First, you can create by yourself if it is simple or not a usual hand.

We can also use the technical analysis library of Python (ta). In this part, we will create the needed indicators by ourselves.

For this example, we have created a mean of returns and volatility of returns. These indicators cannot be found in the ta library so, we need to compute ourselves. Moreover, we have put two timeframes on each indicator.

Code 9.1: Features engineering

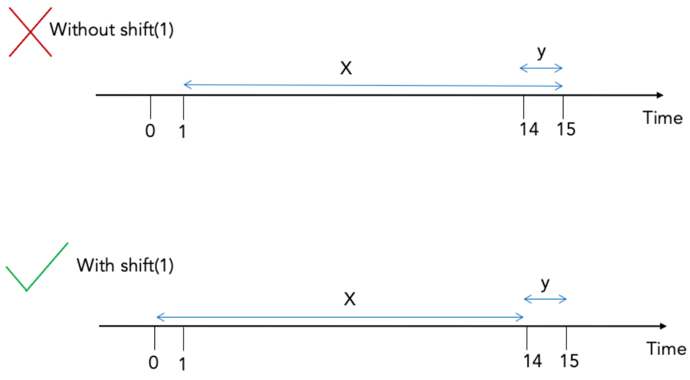
```
# Features engeeniring
df["returns t-1"] = df[["returns"]].shift(1) (1)


# Mean of returns
df["mean returns 15"] = df[["returns"]].rolling(15).mean().shift(1)
df["mean returns 60"] = df[["returns"]].rolling(60).mean().shift(1)


# Volatility of returns
df["volatility returns 15"] = df[["returns"]].rolling(15).std()
.shift(1)
df["volatility returns 60"] = df[["returns"]].rolling(60).std()
.shift(1)
```

(1) It is common as this step because we have seen it many times but, the shift to the feature columns allows us to put the X and the y at the same day at the same row.

Book Link : https://www.amazon.com/gp/product/B09HG18CYL

Do not forget to add a shift to these indicators to not interfere in the data. Let us see why in figure 9.1

Figure 9.1: How to avoid interferences in the data



This figure shows that if you do not shift the data, you will have interferences because you predict the 15th day while you already have the 15th day in the features. It cannot be accurate, and if you make this type of error, you can lose a lot of money.

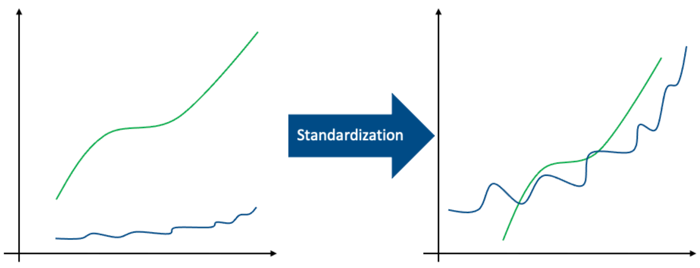
9.1.2. Standardization

In this part, we are going to explain the concept of standardization. We see this in the SVM chapter because it is a geometric algorithm, and it is necessary to standardize the data for these types of algorithms. Still, you can apply this method to another algorithm.

The standardization allows doing the computation faster. Thus, it is interesting for the algorithm which demands many resources.

Sometimes like with this dataset, the data are not at the same scale. For example, we can have a volatility of 60% instead of the returns of 1.75%. So, the algorithm has a lot of difficulty working with that. You need to standardize the data to put it on the same scale. Let us see graphicly why we need to standardize the data in figure 9.2.

Figure 9.2: How standardization works



In this figure, we can see that it is difficult for us to understand the pattern between the two lines before standardization. It is also for the algorithm. Thus, we need to standardize the data to help us to understand it better.

Now, let's see the formula to standardize the data and standardize the data using Python.

$$z_i = \frac{x_i - \mu_x}{\sigma_x}$$

Where z_i is the standardize value, x_i is the value of the observation, μ_x is the mean of the vector x and σ_x the volatility of the vector x.

Code 9.2: Standardize the data

```python
# Import the class
from sklearn.preprocessing import StandardScaler


# Initialize the class
sc = StandardScaler()


# Standardize the data
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

Book Link : https://www.amazon.com/gp/product/B09HG18CYL

As for the other algorithms, we need to fit on the train set only because we cannot know the test set's mean and standard deviation in real life.

I never standardize the y because it does not intervene in the calculations. It is only used to calculate the error of the model except for some specific models.

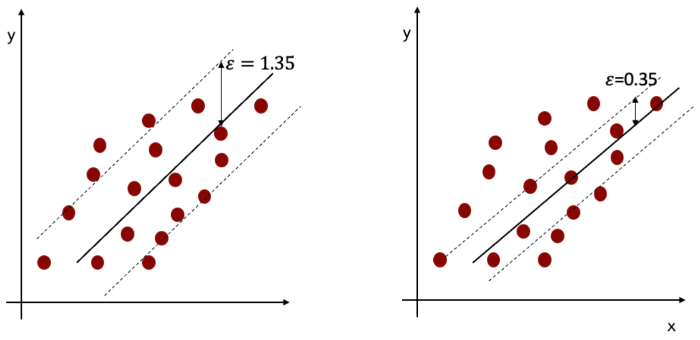
9.3. Support Vector Machine Regressor (SVR)

In this section, we will explain the intuition of the Support Vector Machine Regressor (SVR), then create the model and make a trading strategy using the prediction of our SVR.

9.3.1. Intuition about how works an SVR

In this part, we will learn how an SVR works. It will be easy to understand because the SVR follows nearly the same process as the SVC. We need to take the problem in another way. Instead of maximizing the distance between two groups, the SVR tries to maximize the number of observations between the margins or minimize the number of observations outside the margins. Let us see an example in figure 9.7.

Figure 9.7: Intuition about SVR



In this figure, we can see the functioning intuition of the SVR. The most useful parameter is the epsilon because it manages the path of the SVR.

The parameter epsilon is the tolerance of the model. It can be changed using the hyperparameter of the scikit-learn function. Many functions allow you to find the best combination between the hyperparameters of the models (we will see a technic in the next chapter), but do not forget that the more you optimize the algorithm to fit our data, the more the risk of overfitting increases.