Introduction to Machine Learning

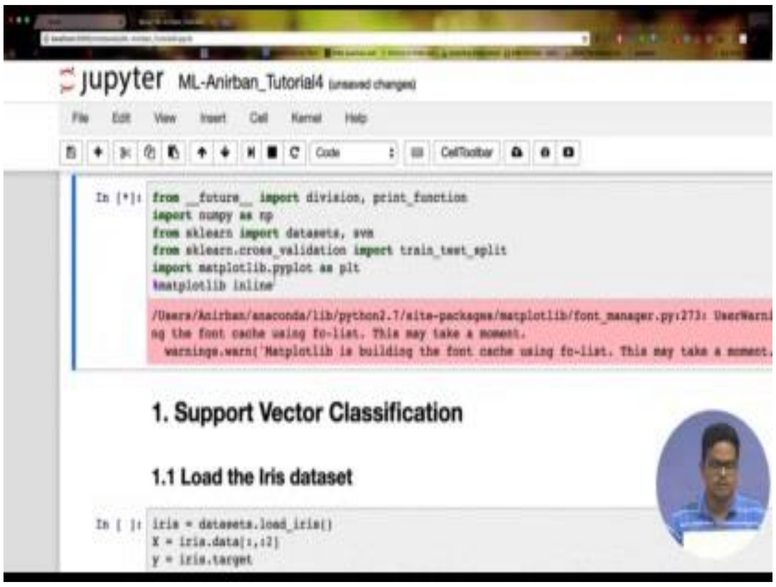
Prof. Mr. Anirban Santara
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 26
Python Exercise on SVM

Hello friends, Anirban here. Welcome to the hands on python coding session of the 5th week of this course. Today we are going to learn how to work with support vector machines in Scikit Learn. So, we will take up support vector classification. First, see how different kernels affect, affect the performance of the support machine classifier and then we will have a demo, a quick demo, a quick glance at support vector regression, because support vector regression is not a part of this course still like as we are doing classification. We are also going to do regression.

So, the datasets that we are going to use today are the Iris dataset, which we have used almost in all of the coding sessions and the Boston house price prediction dataset, which we will be using in the regression part and will be predicting the price of houses from certain features. And this session is going to be a bit different from the previous coding sessions because this time I am going to execute the code directly from the IPython notebook cell by cell and you will be like able to, like follow the flow of the code better. So, let us see how it goes.

(Refer Slide Time: 01:31)


jupyter ML-Anirban_Tutorial4 (unsaved changes)
File Edit View Insert Cell Kernel Help
Code
CellToolbar

```
In [*]: from __future__ import division, print_function
        import numpy as np
        from sklearn import datasets, svm
        from sklearn.cross_validation import train_test_split
        import matplotlib.pyplot as plt
        %matplotlib inline

/Users/Anirban/anaconda/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserWarni
ng the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.
```

1. Support Vector Classification

1.1 Load the Iris dataset

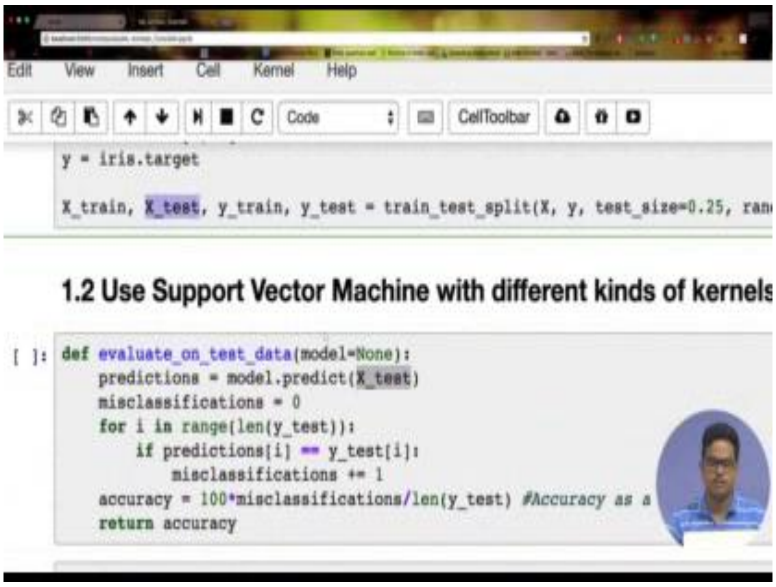
```
In [ ]: iris = datasets.load_iris()
        X = iris.data[:,:2]
        y = iris.target
```

So, the first thing that I want to tell you people is a quick recap of what support vector machine really is. So, support vector machine is a linear classifier. So, the entire idea is about having, about like finding the linear decision boundary, which is at the maximum margin from the training point. So, what is the margin? The margin of a, the margin is defined as the perpendicular distance of a point from the decision boundary.

So, the support vector machine classifier tries to maximize the minimum margin of the training points. It tries to like fit our decision boundary, which tries to generalize as much as possible and the amount of trade off that we want to make for the training set accuracy and you know margin or the amount of test set generalization is controlled by the regularization parameter. All of these things have already been taught in the theory and the tutorial sessions and today we are going to see how they can be implemented in Scikit Learn.

So, first we do all the imports. The modules which we will be using are like NumPy, and then we will be using the datasets of Scikit Learn and the SVM module. And also, we are going to use the train test split function from Scikit Learn dot cross validation and then, we import matplotlib and say, that like we want to use inline plots. So, (Refer Time: 03:25) the first code, the first block of code. So, all the imports are done now, cool.

(Refer Slide Time: 03:30)



```
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ran
```

1.2 Use Support Vector Machine with different kinds of kernels

```
[ ]: def evaluate_on_test_data(model=None):
         predictions = model.predict(X_test)
         misclassifications = 0
         for i in range(len(y_test)):
             if predictions[i] == y_test[i]:
                 misclassifications += 1
         accuracy = 100*misclassifications/len(y_test) #Accuracy as a
         return accuracy
```

So, now we first load the dataset. So, the dataset that we are going to use is the Iris dataset and the dataset dot load Iris. This function directly loads the dataset into the Iris variable and next, we load the features, the input features into the variable x and the input labels into the variable y, and then we split the entire dataset into training and test sets. And we use the trained test split function from Scikit Learn cross validation and we say that the test size will be just 25 percent of the total example size, cool. So, we go ahead and execute this done.

Now, we are going to, you know, check the performance of different kinds of kernels on, at the problem of classification of Iris (Refer Time: 04:31). So, we define a function, which we name, like evaluate on the test data. So, it is going to take a module and it is going to evaluate it on the test data. So, first we are going to find the predictions, alright, the predicted values from the model as for the inputs given.

So, all the test inputs, they are to be, they are going to be supplied through the variable X test. So, X test you can see, that it has been created here, right, by the train test split. So, it is going to take the test, that inputs and it is going to predict their corresponding labels. And now we declare a variable, which is called misclassification, which is going to calculate how many counts, how many misclassifications has, have happened and next we go ahead and we do run this for loop. So, in this for loop we are looping over the entire length of the test set and, and if the prediction, if the ith prediction is not is equal to the actual prediction, then, oops, so there is a, there is a bug. So, let me fix it.

So, it is not misclassification, it is correct classification, alright. So, number of correct classification, we initialize it as 0 and every time if we find, when we find, that the prediction is equal to the actual value, the target value. Then, we increase the number of correct classifications by 1 and then finally, we calculate the accuracy as the number of correct classifications. So, this has already calculated the total number of correct classifications and length of y test will give you the total number of test cases. So, this fraction times 100 will give us the accuracy as a percentage and so, we have the accuracy as a percentage over here and we return the accuracy.

And next, we compile this function and then, we, we are going to explore three kinds of kernels here, linear, polynomial and RBF kernels and RBF kernels are also happened to be called as Gaussian kernel. So, we have a new declarer, new array or a list as you say in Python, called accuracies and you say, that for like index and kernel, in enumerate kernel. So, enumerate kernel is going to return an index, which will be like in z i, the 0 or 1 or 2 and the corresponding element from this tuple. So, for index and kernel and enumerate kernels and so, like you are going to take each kernel one by one and then, you first declare the model.

So, svm.SVC, as we can see over here, svm.SVC. So, svc stands for support vector classifier and svm.SVC is going to make an instance of the classifier with the value of the kernel equal to the kernel that we have like we are going to take in this particular in a particular iteration. So, first linear is going to come and sit here, then polynomial, then RBF. So, we are going to take the kernels one by one in this for loop and then fix it on the training data.

So, model.fit as you have seen earlier, so there is like Scikit Learn has this wonderful, like really, really impressive API in which every single machinery model has the same API. You always instantiate the model with certain parameters and hyper parameters, with the hyper parameter values and then, you just do a model.fit and on the training data, and model dot predict on the test data. And so, this is really impressive and this is really nice.

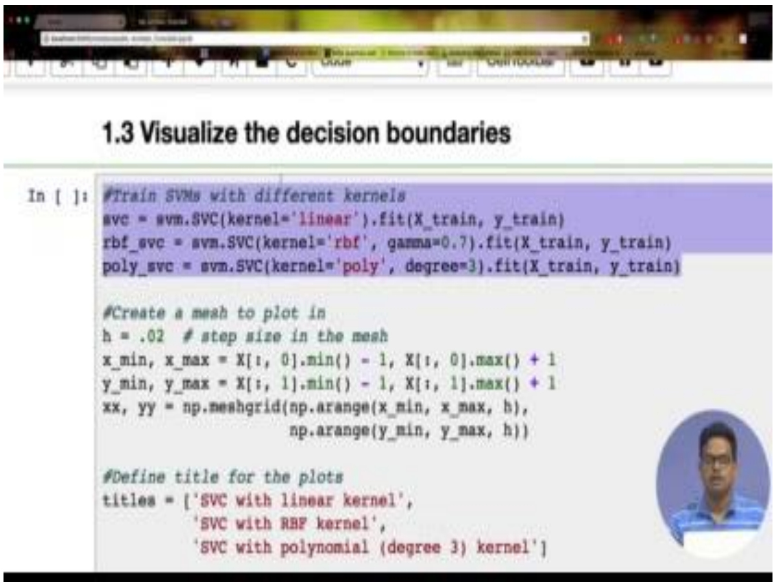
And the same thing, as you have seen before, is going to like we applied as well here, so model.fit x train y train does the training of the SVM and then finally, we calculate the accuracies. So, we invoke this function, which we have declared before, we have defined before and we pass the trained model and ask it to evaluate it on the test data and it returns the accuracy and you just accumulate the accuracies in this accuracy vector, alright. So, you just append the acc, the current accuracy and the accuracies list and then finally.

So, this is, if you do not know this particular syntax, so this syntax is that of the print command which is in Python 3 and I have used this for a purpose over here and so, to activate this particular syntax of print command you have to like import from this underscore underscore future underscore underscore. You have to import this print function. So, if you do this and you have to do this before all other previous imports.

So, if you do this import, then the syntax of print function changes and this particular print function is much more versatile and much more elegant than the one that was used in used in Python 2. So, to use this particular syntax in Python 2, you have to do the import from future and then, this goes like this. So, this is going to be, so this is a blank space holder, it is going to be filled by accuracy. So, much of accuracy has been obtained with kernel this.

So, we are going to, like, test every single kernel one by one in this particular section. So, let us go ahead and run this, cool. So, here we have the results and see that the accuracies of the linear and RBF kernels are equal. And this is the particular case over here not always is the same case and the accuracy of the polynomial kernel is little bit lower.

(Refer Slide Time: 10:47)



1.3 Visualize the decision boundaries

```
In [ ]: #Train SVMs with different kernels
        svc = svm.SVC(kernel='linear').fit(X_train, y_train)
        rbf_svc = svm.SVC(kernel='rbf', gamma=0.7).fit(X_train, y_train)
        poly_svc = svm.SVC(kernel='poly', degree=3).fit(X_train, y_train)

        #Create a mesh to plot in
        h = .02  # step size in the mesh
        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))

        #Define title for the plots
        titles = ['SVC with linear kernel',
                  'SVC with RBF kernel',
                  'SVC with polynomial (degree 3) kernel']
```

So, let us now go ahead and try and visualize the decision boundaries. So, this will give a good insight into how the problem is being solved by the support vector classifier. So, let us just go ahead and look at the code. So, this particular section visualizes the decision boundaries. So, the first segment we are going to train different kinds of classifier so that we train them all over once more. So, we just name them as SVC.

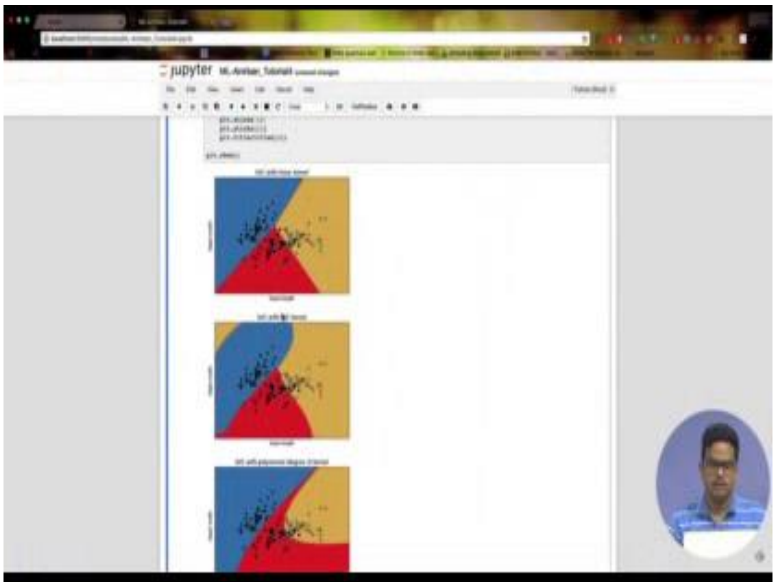
So, the simple svc, this model is going to be a linear kernel SVM and it is it has been trained on the training data. The second RBF SVC is trained is an RBF kernel SVM and it has been trained. So, gamma is another parameter of the sum. So, you can actually look up the documentation of support vector machine dot support vector classifier of Scikit Learn and you will find all of those, like the definitions of each of these parameters and then, you train it on the training data. And now, you have a polynomial kernel SVM in the same way. So, here you are going to, so in this section I will highlight, so in this section you are going to initialize different kinds of kernels and train them.

Now, we would like have to have a mesh plot, which will look beautiful. So, that is why you are, in this section you create a mesh of points. So, it extends from the minimum possible value of the training set, of each feature of the training set to their maximum possible values. So, x min and x max have been calculated, right, and a bit, and like one point earlier and one point later. So, this code is just for like nice plot and then you define the mesh using NumPy dot meshgrid.

And now, you go ahead and discuss and we are making plots. So, the titles will be like this. So, you are making an array of these titles and then, you again, like, go ahead and evaluate different modules. So, it is like in this particular line what we are going to do is, we are again enumerating over this tuple. So, all of these three models separated by commas within first brackets make a tuple, which is iterable and when you use enumerate on this, then it returns an index of each element. It returns each element one by one, returns the element one by one. It returns in the index of the element as well as the element itself.

So, we are going to have svc first with an index of 0, i equal to 0 and then, RBF SVC and then, poly SVC and then, you first initialize the figure and you do all the predict for all the points in the mesh. So, what we are trying to do over here is find out which part of the space belongs to which particular category and you will see in a second how this thing actually looks. So, then you put everything into a contour plot and finally, place the points of the dataset in the plot and do the labeling of the axis over here and then do some more cosmetic changes to the plot and then, you put the titles on the plots. And finally, you show all the plots. So, let us go ahead and run this part.

(Refer Slide Time: 14:31)



See how beautiful it is. So, it generates these three plots and the 1st one is for the linear kernel, the 2nd one is for the RBF kernel and 3rd one is for the polynomial kernel. So, let us go ahead and inspect these one by one and compare. So, in the linear kernel, linear kernel is nothing, but just a linear decision module, right, like no future transmissions. The entire idea of kernels is like this. You have this set of, like, you may have a set of training points.

As SVM is a linear classifier, it always makes a linear decision boundary. When you have a dataset in which the points are not linearly separable, then it does not perform well, right. So, you need to have some non-linear decision boundary and that is why, that is why, kernels come in. So, the idea is that when the points are not linearly separable, you are going to transform them using some feature transformation function into another higher dimensional space preferably in which the points would presumably be linearly separable.

So, you are assuming that the points are not linearly separable in the current feature space, but when they are mapped to a much higher dimensional feature space, then they will be linearly separable. So, you first do that feature mapping using some non-linear function and then, in that higher dimensional space you are, you will go ahead and make an SVM, right. This is the linear decision boundary, but this entire process can be bypassed. So, first feature transformation, then SVM. So, this can be bypassed for using a kernel.

So, kernel function is a function which returns directly the dot product of the transformed points from the points, in the initial, of the original feature space. So, a kernel function takes points from the original feature space and returns the dot product of those points in the transformed space and I have explained this more explicitly in the, with hands, with proper figures and everything in the tutorial session of this week.

So, entire idea of kernel is to extrapolate SVMs to the non-linear decision boundaries, to have a SVM with non-linear decision boundaries. So, you do not, you continue to have a linear decision boundary, but the decision boundary is in a different space, so to which you first transform your data points to, right. So, this linear kernel SVM is doing the, you know, linear decision boundary classification in the original feature space.

And you can see that as the points of the first class, this green point, they are linearly separable from the other classes, so they have been neatly separated into a different region. However, the other two classes, they are not linearly separable and hence, you need, like this linear decision boundary, it tries to like, you know, maximally separate the points of the two classes, but it has a lot of misclassifications, right. However, see what happens with the RBF kernel.

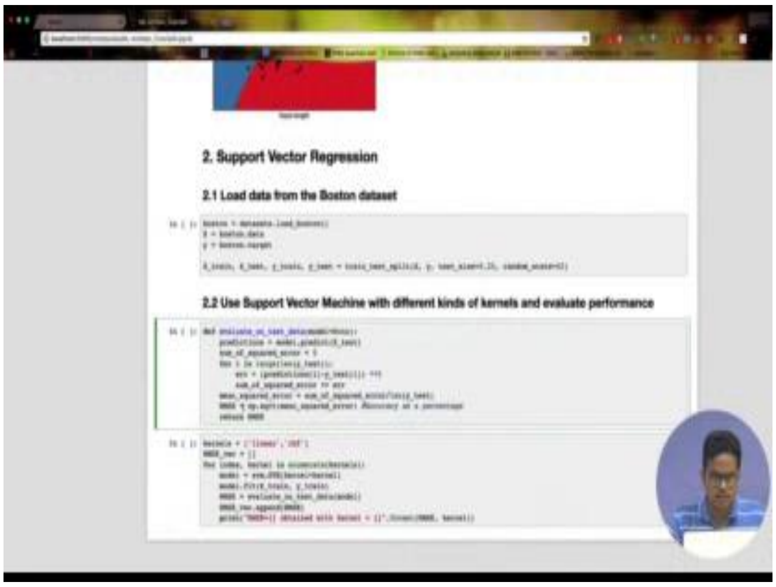
So, all of these decision boundaries, they are non-linear. As you can see, the linear decision, the linearly separable first class, continues to be separated neatly, but in this case, the points of the, white points and these blue points have been separated better because as you can see before, that the accuracy, oops, this was for the polynomial kernel, yeah. So, it does not, it does not, so the RBF kernel gives the same accuracy as the linear kernel for this particular example. However, you can see that the decision boundaries are non-linear now. So, you could have, as these decision boundaries are non-linear, so you could have a better performance in this case.

And the only thing that I want you people to notice here is that the decision boundaries become non-linear. So, the points are first transformed using the, using a particular feature transformation function and then, in that transformed domain they are separated by linear decision boundaries and when those decision boundaries are visualized in the original space, they look non-linear.

And similar is the case with polynomial kernel and you can see, that it is like, it has tried to divide the space using like, you know, cubic polynomials. So, as it is a degree three kernel, so it will be fitting polynomials of degree three and splitting the space, whereas these would look like, you know, contours in Gaussian curves, like Gaussian curves.

So, if you like, it is like, it has like fitted Gaussian on the data, one on this part one, on this and one on that, or maybe I am not sure about how many components have been chosen, but you will find out that there is a function and these things, it is like, you know, like these decision boundaries, they are contours of a Gaussian. So, this is how it is and both of these, like polynomial kernel and the RBF kernel, they give non-linear decision boundaries, whereas the linear kernel gives linear decision boundaries.

(Refer Slide Time: 20:24)



2. Support Vector Regression

2.1 Load data from the Boston dataset

```
In [ ]: boston = datasets.load_boston()
        X = boston.data
        y = boston.target

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

2.2 Use Support Vector Machine with different kinds of kernels and evaluate performance

```
In [ ]: def evaluate_on_test_data(model=None):
            predictions = model.predict(X_test)
            sum_of_squared_error = 0
            for i in range(len(y_test)):
                err = (predictions[i]-y_test[i]) **2
                sum_of_squared_error += err
            mean_squared_error = sum_of_squared_error/len(y_test)
            RMSE = np.sqrt(mean_squared_error) #Accuracy as a percentage
            return RMSE
```

```
In [ ]: kernels = ['linear','rbf']
        RMSE_vec = []
        for index, kernel in enumerate(kernels):
            model = svm.SVR(kernel=kernel)
            model.fit(X_train, y_train)
            RMSE = evaluate_on_test_data(model)
            RMSE_vec.append(RMSE)
            print("RMSE={} obtained with kernel = {}".format(RMSE, kernel))
```

So, we can like go ahead and find see how support vector regression works. So, support vector regression works this way. So, first we load the data. So, datasets dot load boston is a function that loads the boston house price prediction datasets of Scikit Learn into the environment and stores it within this variable boston. We take out the inputs and the targets, make the training and test splits and then, we have a similar evaluate on test data function. And over here, as this is a regression problem, as this is a regression problem, we do not have classification error, misclassification as our objective.

So, what we are going to use is the mean squared error, the root mean squared error. So, we calculate it this way. So, first we initialize the sum of squared error, this variable as 0 and then we calculate the square of the error. So, whatever number has been predicted by the system, so model dot predict x test will give the predictions for the test data and each prediction predicted value is subtracted from the true value and raised to the power of 2.

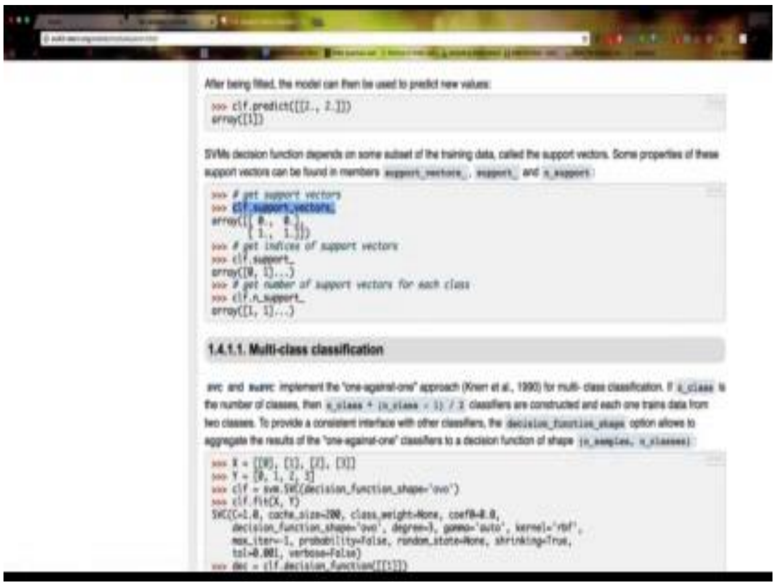
So, this gives the squared error. We keep accumulating that within this variable and we get the sum of squared error and then finally, we calculate the mean squared error by dividing the sum of squared error by total number of points in the test set and then, oops, this should not be accuracy, let me remove this comment. So, the root mean squared error is given by np dot squared root. So, you find the square root of the error of the mean squared error and you have root mean squared error.

So, this gives an estimate of how close the or how far away the predicted values from the regressor are from the true values and let us evaluate just two kernels over here. So, first we compile this part, and go ahead to the next part. So, we will evaluate the linear kernel SVM and the RBF kernel SVM in this section. Let me zoom down a little bit, okay, it is better now.

So, in the RMSE vector, this thing is going to accumulate the root mean squared error values and first we will, we will, just like we previously we did. So, we take the kernels one by one, we fit the support vector depressor onto the, with the kernel on the training data and then, we calculate the root mean squared error by calling this function and we append this value to the RMSE effect and finally, print these values out. So, let us see how these work. So, we find that the RMSEs are 0.3774 and 0.34. So, in this case as you can see, that the RBF kernels works better and because it can fit more, it can, it can fit non-linear curve onto the data.

And there are some more things that you can do with these models. For example, it can actually find out all the different. So, I will insert us one more cell below. So, I will show you something more, something else. So, let us take one of these classifiers, one of these models. So, we take the RBF SVC and we find out what all points were used as support vectors by these classifiers.

(Refer Slide Time: 25:47)



After being fitted, the model can then be used to predict new values:

```
>>> clf.predict([[2., 2.]])
array([1])
```

SVMs decision function depends on some subset of the training data, called the support vectors. Some properties of these support vectors can be found in members `support_vectors_`, `support_` and `n_support`:

```
>>> # get support vectors
>>> clf.support_vectors_
array([[ 0.,  0.],
       [ 1.,  1.]])
>>> # get indices of support vectors
>>> clf.support_
array([0, 1]...)
>>> # get number of support vectors for each class
>>> clf.n_support_
array([1, 1]...)
```

1.4.1.1. Multi-class classification

`SVC` and `NuSVC` implement the "one-against-one" approach (Knerr et al., 1990) for multi- class classification. If `n_class` is the number of classes, then `n_class * (n_class - 1) / 2` classifiers are constructed and each one trains data from two classes. To provide a consistent interface with other classifiers, the `decision_function_shape` option allows to aggregate the results of the "one-against-one" classifiers to a decision function of shape `(n_samples, n_classes)`:

```
>>> X = [[0], [1], [2], [3]]
>>> Y = [0, 1, 2, 3]
>>> clf = svm.SVC(decision_function_shape='ovo')
>>> clf.fit(X, Y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
>>> dec = clf.decision_function([[1]])
```

So, I will open the Scikit Kearn documentation, Scikit Learn. So, this documentation gives you a lot of resources. Something is wrong; yes there we go, yes. So, this particular attribute of a model, the support vectors underscore will give you the support vector. So, if we can go ahead and if you want to see what all points were used as support vectors, we can bring this here. So, we see can that so many support vectors.

So, all of these points were used as support vectors and if we see if we check how many support vectors were used here, we can just find the length of this array and there are 70 support vectors for the RDF kernel SVM. Whereas, if you just use the svc, if you check for the svc, there are 67 support vectors and they can be like found this way, just like remove the length, you will see all of those points that were used as support vectors.

So, these are all the points from, all of these points are from the training side and these are the points, which define the decision boundary that the classifier is using and the polynomial one. So, poly svc is using, length of poly svc support vectors, if we can check there are 55 support vectors over here. So, these things and there are other attributes as well and a lot of attributes and the documentation of Scikit Learn is really good and all of these parameters.

So, I will just open the, let me open, linear svc, just svc, yes. So, these are the different parameters of the model and you can have different combinations of them according to the need of the problem at hand like these things, these are the different, we use the support vectors. All of these different parameters of the support vector machine can be obtained by just invoking the corresponding attribute value.

So, that is all for today. Today, we studied how to use the support vector machine module of Scikit Learn to have classifications and regressions and there are a lot of, like support vector machines are extremely sophisticated and extremely efficient machine learning models, and they have been the state of the art at a lot of different applications for a long time. And I would like to recommend you to go ahead and find other applications of support vector machines as well and try to understand, learn, learn, understand and you know, implement this particular part of this course that is the support vector machines and kernel machines and all of this theory very well. So, this is one of the most important sections of this course.

See you next time, bye-bye.