```
# First Task :


from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plottt
from sklearn.linear_model import LinearRegression
import numpy as  numpie
from sklearn.metrics import mean_squared_error

# A Function to Generate Data
def toydata_generate(f, sizeofthesample, varation_of_the_noise):
    x = numpie.linspace(0, 1, sizeofthesample)
    t = f(x) + numpie.random.normal(scale=varation_of_the_noise, size=x.shape)
    return x, t

def f(x):
    return  numpie.sin(2 * numpie.pi * x)

# 1. Produce 100 points for testing and 10 points for training.
train_for_x,train_for_y = toydata_generate(f, 10, 0.25)
test_for_x = numpie.linspace(0, 1, 100)
test_for_y = f(test_for_x)

#2. Use the polynomial basis function (order M=9) in step two.
poly_features = PolynomialFeatures(degree=9)
Train_Phi_Feat = poly_features.fit_transform(train_for_x[:, numpie.newaxis])
Test_Phi_Feat = poly_features.transform(test_for_x[:, numpie.newaxis])

# 3. The model should be trained parametrically and the test MSE should be reported
lr = LinearRegression(fit_intercept=False)
lr.fit(Train_Phi_Feat,train_for_y)
Pred_for_y = lr.predict(Test_Phi_Feat)
Mean_Sqre_Err = mean_squared_error(test_for_y, Pred_for_y)
print("In a parametric test, the mean square error is as follows:", Mean_Sqre_Err)

# 4.Predict non-parametrically
K = Train_Phi_Feat @ Train_Phi_Feat.T
k = Test_Phi_Feat @ Train_Phi_Feat.T
non_paramertic_Pred_for_y= k @ numpie.linalg.inv(K) @train_for_y
non_param_mean_sqaure_error = mean_squared_error(test_for_y,non_paramertic_Pred_for_y)
print("Non-parametric test MSE:", non_param_mean_sqaure_error)

# 5. Compare the predictions
print("What is the similarity between the predictions?", numpie.allclose(Pred_for_y,non_paramertic_Pred_for_y))
print("Hence they were not similar")


# Second Task :

```

```


from sklearn.metrics.pairwise import rbf_kernel
import numpy as numpie
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plottt

#1. Using the RBF kernel, define the gram matrix
gamma = 5
K = rbf_kernel(X=train_for_x[:, numpie.newaxis], Y=train_for_x[:, numpie.newaxis], gamma=gamma)

# Setting Up Beta = 10 for the covariance matrix
beta = 10
C = K + numpie.eye(len(K)) / beta

#2. The invertibility of C should be checked
try:
    C_inv = numpie.linalg.inv(C)
    print("The value of C can be inverted.")
except numpie.linalg.LinAlgError:
    print("The value of C cannot be inverted.")

#3. All test samples should be calculated to determine the predictive mean
Test_k = rbf_kernel(X=test_for_x[:, numpie.newaxis], Y=train_for_x[:, numpie.newaxis], gamma=gamma)
Gauusian_Prediction_Pred_for_y = Test_k @ C_inv @train_for_y

# Test MSE
Gaussain_Process_Mean_Sqre_err = mean_squared_error(test_for_y, Gauusian_Prediction_Pred_for_y)
print("The Gaussian Process (MSE) should be tested as follows:", Gaussain_Process_Mean_Sqre_err)
# ```

# Third Task :
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

#Separate Datasets of training and Testing  from the iris _ dataset and load them up
iris = load_iris()
Train_for_X, Test_for_X,train_for_y, test_for_y = train_test_split(iris.data, iris.target, test_size=0.3, random_state=5)

# SVC model creation and training
model_svc = SVC()
model_svc.fit(Train_for_X,train_for_y)

# 2.1:Multi-class classification is handled by SVC by default using a one-versus-one approach
print("To classify multi-classes, SVC uses a one-to-one approach.")

# 2.2: Number of SVM's or the support vectors
print("No of SVM's or Support vectors :", len(model_svc.support_))

```

```
# 2.3: Identifying  that the support vector is in the 18th training sample
Is_Supprt_vectr = 18 in model_svc.support_
print("Did a support vector can be found in the 18th training sample?", Is_Supprt_vectr)

# 2.4: Counting No Of SVM"S from Class 2
Class_2_n_Support_Vectors = model_svc.n_support_[2]
print("The no of Support Vector's  class 2 from are :", Class_2_n_Support_Vectors)

# 3: Calculate the accuracy of the classification test
Pred_for_y = model_svc.predict(Test_for_X)
test_accuracy = accuracy_score(test_for_y, Pred_for_y)
print("Calculation of the accuracy of the classification test is that :", test_accuracy)


# I have completed the third task. SVM model classification test accuracy is reported by checking whether the 18th training sample is a support vector, calcul
```

```
In a parametric test, the mean square error is as follows: 0.25065378940033745
Non-parametric test MSE: 0.25071794129118596
What is the similarity between the predictions? False
Hence they were not similar
The value of C can be inverted.
The Gaussian Process (MSE) should be tested as follows: 0.05006472466361181
To classify multi-classes, SVC uses a one-to-one approach.
No of SVM's or Support vectors : 50
Did a support vector can be found in the 18th training sample? True
The no of Support Vector's  class 2 from are : 21
Calculation of the accuracy of the classification test is that : 0.9777777777777777
```

0s completed at 22:01