Code Detailed Description:

Section -D

Part-A: K cross validation(k=5) on 4 Types of SVM’s Kernel

a)Linear kernel

b)Radial Bias Kernel(RBF)

c)Polynomial Kernel

d)Sigmoid Kernel

Data Pre-processing Part: (After importing libraries)

1).Identified the dataset as a 15 Boolean and 2 int columns. The dependent variable is the zoo (in 1 to 7) to which an animal with particular features belongs to.

2).The data file is taken input as csv and the independent variable is assigned values from 1:17 as the 17th column is the dependent variable ,and till 17 all columns are feature vector and 1:17 means including 1st and excluding 17th column.

Command used:

*(‘X=datset.iloc[:,1:17].values’)*

Same for independent variable y we used the 17th column which consists the zoo number.

Command used:

*y=datset.iloc[:,17].values*

3). Then we split the data into the training set and the test set

Using train\_test\_split function of sklearn.model\_selection. And the test size is selected as 25% of the Vectors.

Command used:

*from sklearn.model\_selection import train\_test\_split*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)*

Importing Particular SVMs :

1.)Linear Kernel

We Imported SVC from svm in sklearn and made a classifier object and in kernel parameter we wrote ‘linear’, it initializes a linear kernel to the classifier object.

Command used:

*‘classifier = SVC(kernel = 'linear', random\_state = 0)’*

Then we fit the classifier into the training set and the test set

Command used:

*classifier.fit(X\_train, y\_train)*

Final Stage is to make a prediction vector y\_pred which predicts the values of dependent variable for X\_test.

Command used:

*y\_pred = classifier.predict(X\_test)*

2.) Radial Bias kernel

We Imported SVC from svm in sklearn and made a classifier object and in kernel parameter we wrote ‘rbf’, it initializes a rbf kernel to the classifier object.

Command used:

‘*classifier = SVC(kernel = 'rbf', random\_state = 0)’*

Then we fit the classifier into the training set and the test set

Command used:

*classifier.fit(X\_train, y\_train)*

Final Stage is to make a prediction vector y\_pred which predicts the values of dependent variable for X\_test.

Command used:

*y\_pred = classifier.predict(X\_test)*

3.) Polynomial kernel

We Imported SVC from svm in sklearn and made a classifier object and in kernel parameter we wrote ‘poly’, it initializes a polynomial(degree=3) kernel to the classifier object.

Command used:

*‘classifier = SVC(kernel = 'poly', random\_state = 0)’*

Then we fit the classifier into the training set and the test set

Command used:

*classifier.fit(X\_train, y\_train)*

Final Stage is to make a prediction vector y\_pred which predicts the values of dependent variable for X\_test.

Command used:

*y\_pred = classifier.predict(X\_test)*

4.) Sigmoid kernel

We Imported SVC from svm in sklearn and made a classifier object and in kernel parameter we wrote ‘poly’, it initializes a sigmoid kernel to the classifier object.

Command used:

*‘classifier = SVC(kernel = 'sigmoid', random\_state = 0)’*

Then we fit the classifier into the training set and the test set

Command used:

*classifier.fit(X\_train, y\_train)*

Final Stage is to make a prediction vector y\_pred which predicts the values of dependent variable for X\_test.

Command used:

*y\_pred = classifier.predict(X\_test)*

5.)Using K fold Cross validation

K fold cross validation is separating of the test and the training set into k set (here k=5) and each set of observation is to be fit into the classifier and scores are evaluated and appended in an array of scores of a classifier.

a).First of all we made a function called get\_score which takes the model , the training and the test sets as the inputs to the model and inside the body we fit the training set into it and predicted(returned) the scores.

Code snippet:

*def get\_score(model,X\_train,X\_test,y\_train,y\_test):*

*model.fit(X\_train,y\_train)*

*return model.score(X\_test,y\_test)*

b).Second step was to initialize a k fold object and setting k=5

Code Snippet:

*from sklearn.model\_selection import KFold*

*kf=KFold(n\_splits=5)*

c).Then we divide the indexes using split function and KFold object.

Code Snippet:

*for train\_index,test\_index in kf.split(X\_train,y\_train):*

*x\_train,x\_test=X\_train[train\_index],X\_train[test\_index]*

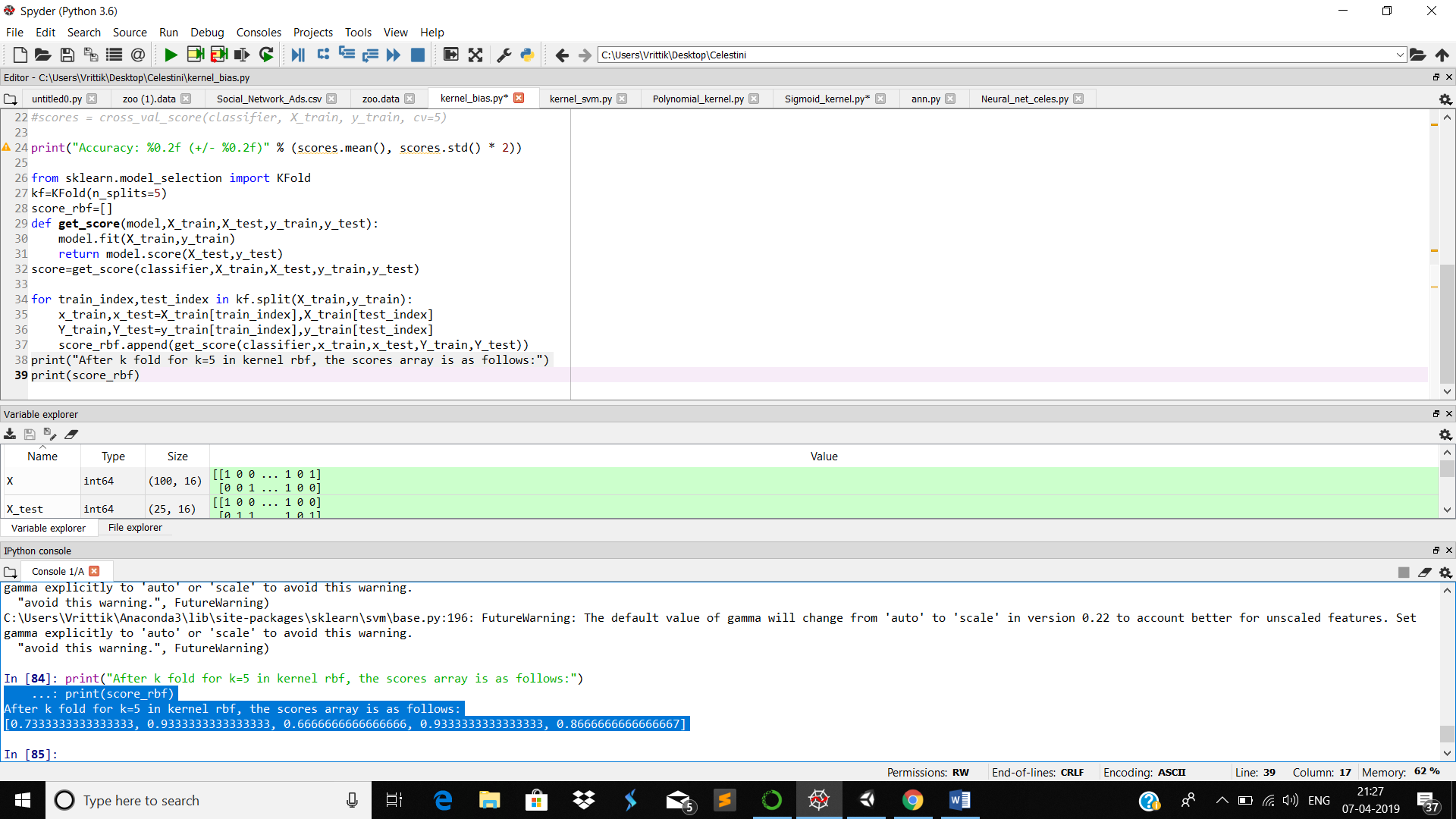
*Y\_train,Y\_test=y\_train[train\_index],y\_train[test\_index]*

*score\_linear\_kernel.append(get\_score(classifier,x\_train,x\_test,Y\_train,Y\_test))*

We did a split into x\_train ,x\_test,Y\_train,Y\_test for each run of the for loop that separates the train sets.

Then we called the get score function and it returns the score of the classifier , we appended all the scores and printed out the scores for the 5 Folds.

One Output example is:



6.)Predicting accuracies

Using the cross\_val\_score of sklearn the scores are predicted as follows:

(‘*scores = cross\_val\_score(classifier, X\_train, y\_train, cv=5)’)*

*Then accuracy is calculated by standard formula:-*

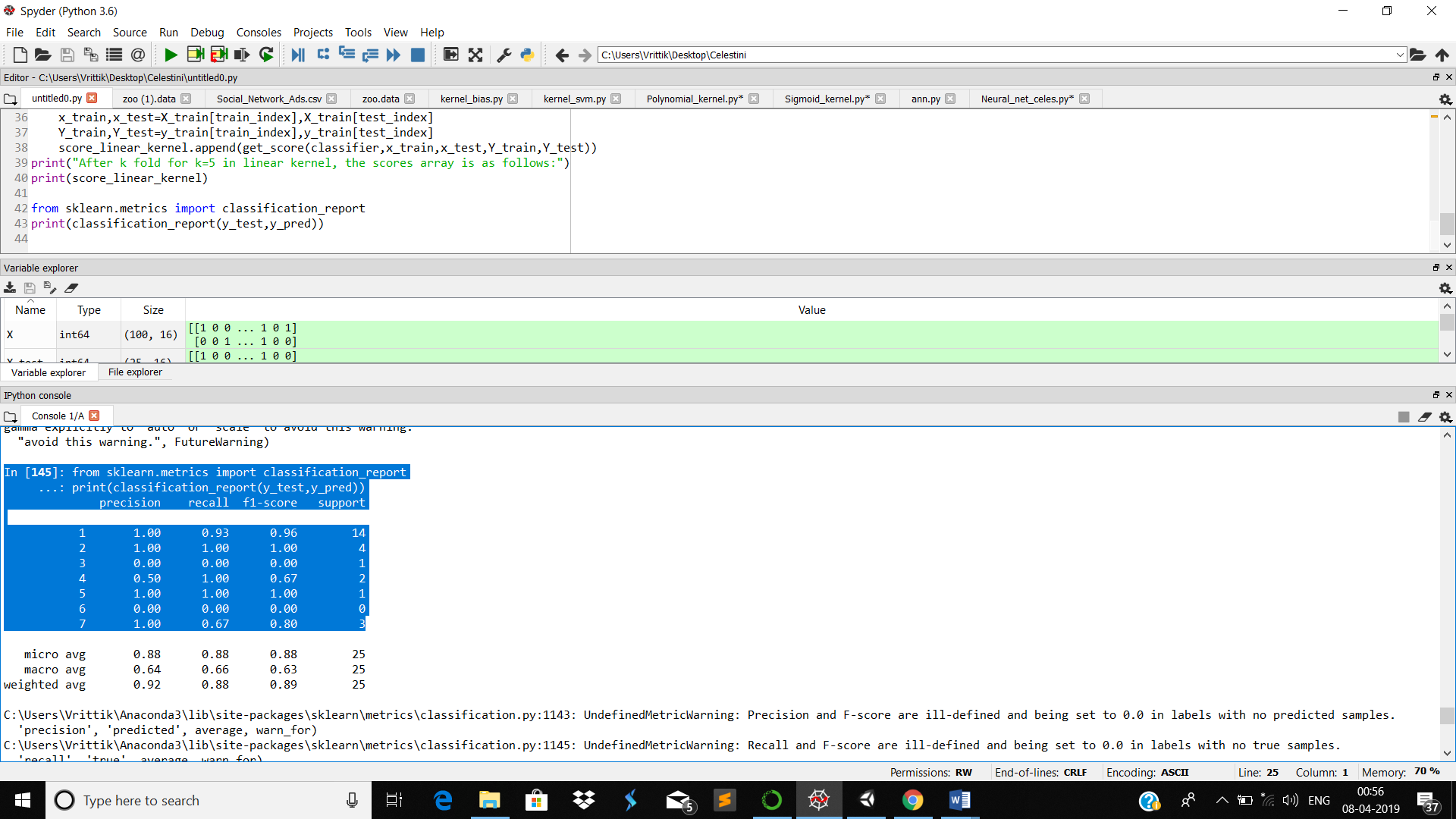
*(‘print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))*’)

7.)calculating the precision and recall

We used sklearn.metrics for importing the classification\_report.

We just y\_true and y\_pred in the argument and a string class of classification report is observed at the output:

Example:



Neural Network:

Data preprocessing:

1.Since this is a multiclass classification problem so we have to use categorical cross entropy for model compilation.

But since the output consist of a column in range(1-7)

2.So we made dummy variable columns representing the independent variable(cause categorical crossentropy needs at least 2 output coloums)

We made dummy variables using the code:

(‘*dummies=pd.get\_dummies(y)*’)

3.Then we appended the dummy variable column into the dataframe and then selected X and y respectively. **One more Thing to notice is that y matrix contain 6 vector not 7 so as to avoid the dummy variable trap.**

(*‘X=final.iloc[:,1:17].values*

*y=final.iloc[:,17:23].values’*)

4.Again we split the dataset into training and the test set

5.Now here comes the making of the neural network layers

We made two hidden layers in the neural network .

In the first layer we took input\_dim=16(as 16 independent variables ) and output\_dim=average of output and inputs(i.e (6+16)/2=11)

And similarly we made the second layer.

The codes are given as:

( *classifier.add(Dense(output\_dim=11, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 16))*

*# Adding the second hidden layer*

*classifier.add(Dense(output\_dim = 11, kernel\_initializer = 'uniform', activation = 'relu'))* )

6.)Optimize layer

We made the backpropagation layer in which we set the ***loss = categorical cross\_entropy*** as there are more than 2 output classes. We set optimizer =adam for optimizing the uniformly initialized connecting layer width.

Code Snippet:

*classifier.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])*

7.)We fit the x\_train and y\_train the model as

Code Snippet:

*classifier.fit(X\_train, y\_train, batch\_size = 10, epochs = 500)*

8) Then we did the same as in the SVM part , I.e predicting for X\_test,storing it in y\_pred and evaluating classification\_report.

Code Snippet:

*from sklearn.metrics import classification\_report*

*print(classification\_report(y\_test,y\_pred))*