Celestini Project 2019

Machine Learning Problem
(Section-B)

Problem Statement : To design a machine learning algorithm for finding the zoo for every animal , given the set of features of the animal.

Models to be used :

1. Multiclass SVM with the following kernels-
   - Linear Kernel
   - RBF Kernel
   - Polynomial Kernel
   - Sigmoid Kernel

2. Back Propagation Neural Network

Task a) Classification Accuracy and Average Precision of each class

- Each of the 4 kernels of SVC
- Back Propagation neural network using cross entropy loss function

Solution : Code Snippet and output

- Linear kernel of SVC

```
# INPUT
scores = []
clf = OneVsRestClassifier(SVC(kernel='linear'))

# Using KFold  for cross validation with n_folds = 5
cv = KFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in cv.split(X):
    X_train, X_test, y_train, y_test = X[train_index], X[test_index], Y[train_index],
Y[test_index]
    clf.fit(X_train, y_train)
    scores.append(clf.score(X_test, y_test))
    y_score = clf.decision_function(X_test)

# Classification Accuracy
print("Classification Accuracy:",np.mean(scores))

# Precision Score
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test[:, i],
                                                        y_score[:, i])
    average_precision[i] = average_precision_score(y_test[:, i], y_score[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.ravel(),
    y_score.ravel())
average_precision["micro"] = average_precision_score(y_test, y_score,
                                                     average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

# **OUTPUT**

```
Classification Accuracy: 0.89
Average precision score, micro-averaged over all classes: 0.90
```

- <u>Radial Basis Function kernel of SVC</u>

```
# INPUT
scores = []
clf = OneVsRestClassifier(SVC(kernel='rbf’))

# Using KFold  for cross validation with n_folds = 5
cv = KFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in cv.split(X):


   X_train, X_test, y_train, y_test = X[train_index], X[test_index], Y[train_index],
Y[test_index]
   clf.fit(X_train, y_train)
   scores.append(clf.score(X_test, y_test))
   y_score = clf.decision_function(X_test)

# Classification Accuracy
print("Classification Accuracy:",np.mean(scores))

# Precision Score
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
   precision[i], recall[i], _ = precision_recall_curve(y_test[:, i],
                                                       y_score[:, i])
   average_precision[i] = average_precision_score(y_test[:, i], y_score[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.ravel(),
   y_score.ravel())
average_precision["micro"] = average_precision_score(y_test, y_score,
                                                     average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
     .format(average_precision["micro"]))
```

```
# OUTPUT
Classification Accuracy: 0.8104761904761905
```

```
Average precision score, micro-averaged over all classes: 0.87
```

- <u>Polynomial kernel of SVC</u>

```
# INPUT
scores = []
clf = OneVsRestClassifier(SVC(kernel='poly'))

# Using KFold  for cross validation with n_folds = 5
cv = KFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in cv.split(X):


    X_train, X_test, y_train, y_test = X[train_index], X[test_index], Y[train_index],
Y[test_index]
    clf.fit(X_train, y_train)
    scores.append(clf.score(X_test, y_test))
    y_score = clf.decision_function(X_test)

# Classification Accuracy
print("Classification Accuracy:",np.mean(scores))

# Precision Score
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test[:, i],
                                                        y_score[:, i])
    average_precision[i] = average_precision_score(y_test[:, i], y_score[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.ravel(),
    y_score.ravel())
average_precision["micro"] = average_precision_score(y_test, y_score,
                                                     average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

# **OUTPUT**

```
Classification Accuracy: 0.6328571428571428
Average precision score, micro-averaged over all classes: 0.72
```

- Sigmoid kernel of SVC

# **INPUT**

```
scores = []
clf = OneVsRestClassifier(SVC(kernel='sigmoid'))

# Using KFold  for cross validation with n_folds = 5
cv = KFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in cv.split(X):


   X_train, X_test, y_train, y_test = X[train_index], X[test_index], Y[train_index],
Y[test_index]
   clf.fit(X_train, y_train)
   scores.append(clf.score(X_test, y_test))
   y_score = clf.decision_function(X_test)

# Classification Accuracy
print("Classification Accuracy:",np.mean(scores))

# Precision Score
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
   precision[i], recall[i], _ = precision_recall_curve(y_test[:, i],
                                                       y_score[:, i])
   average_precision[i] = average_precision_score(y_test[:, i], y_score[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.ravel(),
    y_score.ravel())
average_precision["micro"] = average_precision_score(y_test, y_score,
                                                    average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

# OUTPUT

```
Classification Accuracy: 0.5238095238095238
Average precision score, micro-averaged over all classes: 0.38
```

- Explanation and Inference : Linear kernel gives the best accuracy and average precision. Micro-average precision is taken to handle imbalance in the classes.

  - BPNN

  # INPUT

```
 with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(5001):
        sess.run(train, feed_dict={X: train_x, Y: train_y})
        if step % 1000 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={X:
train_x, Y: train_y})
            print("Step: {:5}\tLoss: {:.3f}\tAcc:
{:.2%}".format(step, loss, acc))

    train_acc = sess.run(accuracy, feed_dict={X: train_x, Y:
train_y})
    test_acc,test_predict,test_correct =
sess.run([accuracy,prediction,correct_prediction], feed_dict={X:
test_x, Y: test_y})
```

```
    print("Model Prediction =", train_acc)
    print("Test Prediction =", test_acc)

# OUTPUT

Step:     0 Loss: 3.402 Acc: 30.00%
Step:  1000 Loss: 0.135 Acc: 87.14%
Step:  2000 Loss: 0.076 Acc: 90.00%
Step:  3000 Loss: 0.054 Acc: 90.00%
Step:  4000 Loss: 0.043 Acc: 90.00%
Step:  5000 Loss: 0.037 Acc: 90.00%
Model Prediction = 0.9
Test Prediction = 0.9354839
```

- Explanation and Inference : This neural network with 7 hidden units ,softmax activation function, cross-entropy loss, and simply divided into training and testing data (instead of K Fold method used in SVM), tends to give good training as well as testing accuracy , showing that there is no underfitting nor overfitting.

Task b) Explanation of the results obtained by the two models

Solution:

SVC-

- Performing K-Fold cross validation on Support vector classifier made the algorithms overfit the dataset.
- This can be seen by low classification accuracy of SVM kernels as compared to neural network
- Comparing the 4 kernels of SVC, linear kernel tends to give best accuracy and average precision score.This is because the data is linearly separable and thus RBF or any other kernel doesn’t provide higher accuracy. Rather they are more expensive computation vise.
- The micro-averaged precision score is taken instead of precion score since it is a multiclass classification problem and micro-average is preferred if there is any possibility of imbalanced classes.

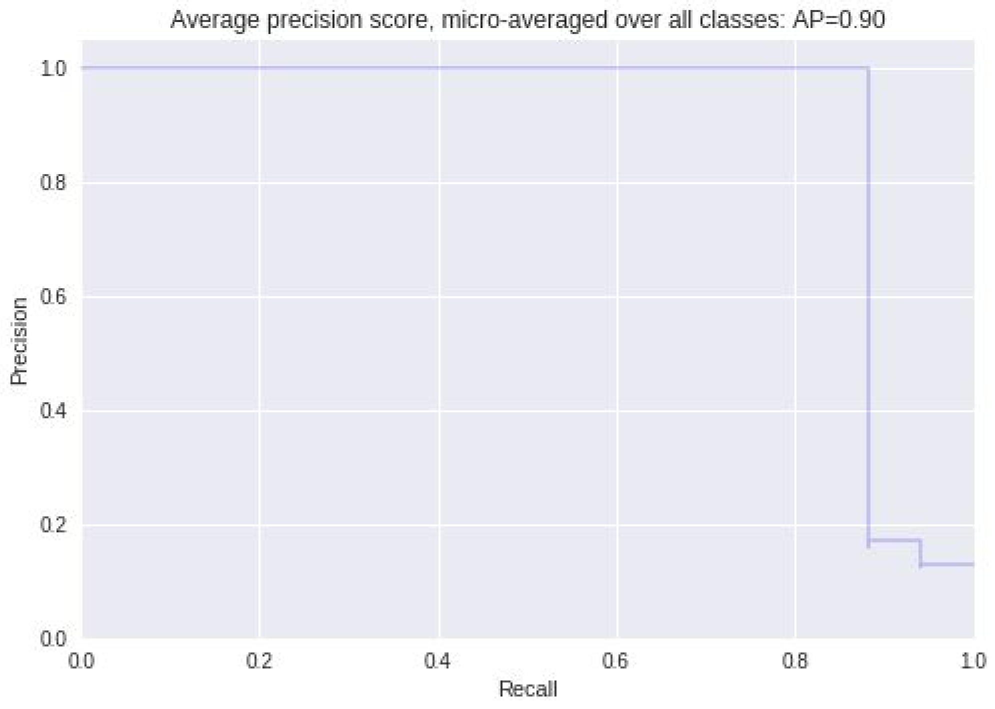
BPNN-

- The neural network is trained using different optimizers like Gradient Descent and Adam.
- The activation function used is 'softmax' and loss function used is 'cross-entropy-loss'.
- For Gradient Descent algorithm , the training accuracy :0.90 and test accuracy : 0.93.
- This shows that there is no overfitting or underfitting in the model.
- The model gives higher classification accuracy as compared to even the best kernel of SVC (that is linear kernel in this case).

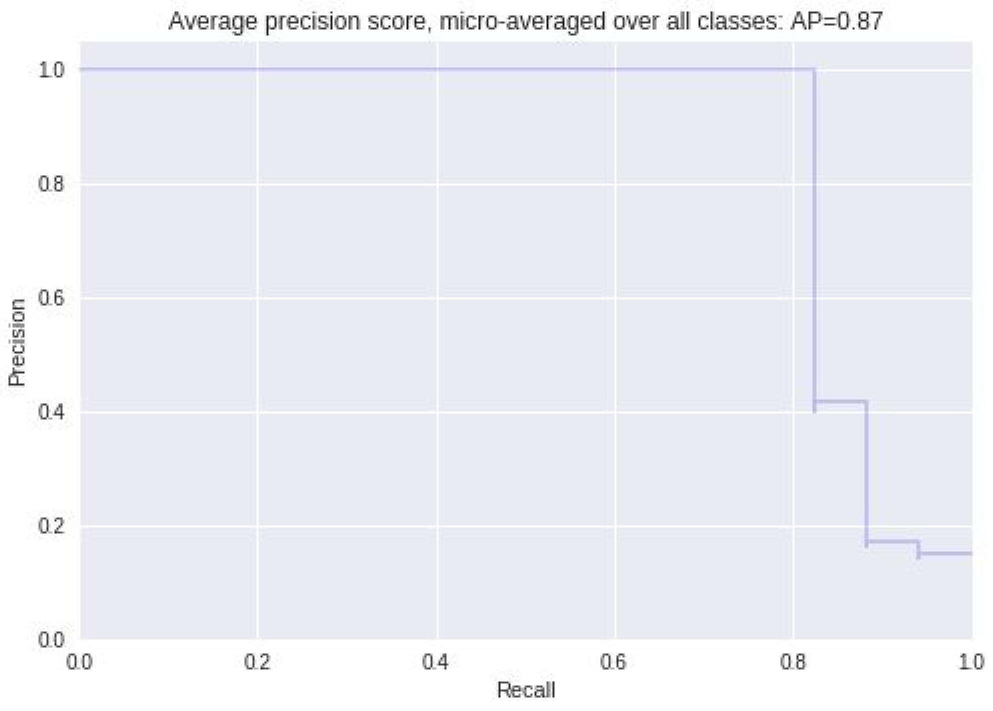
Task c) Plot Precision-Recall Curve for the models

Solution:

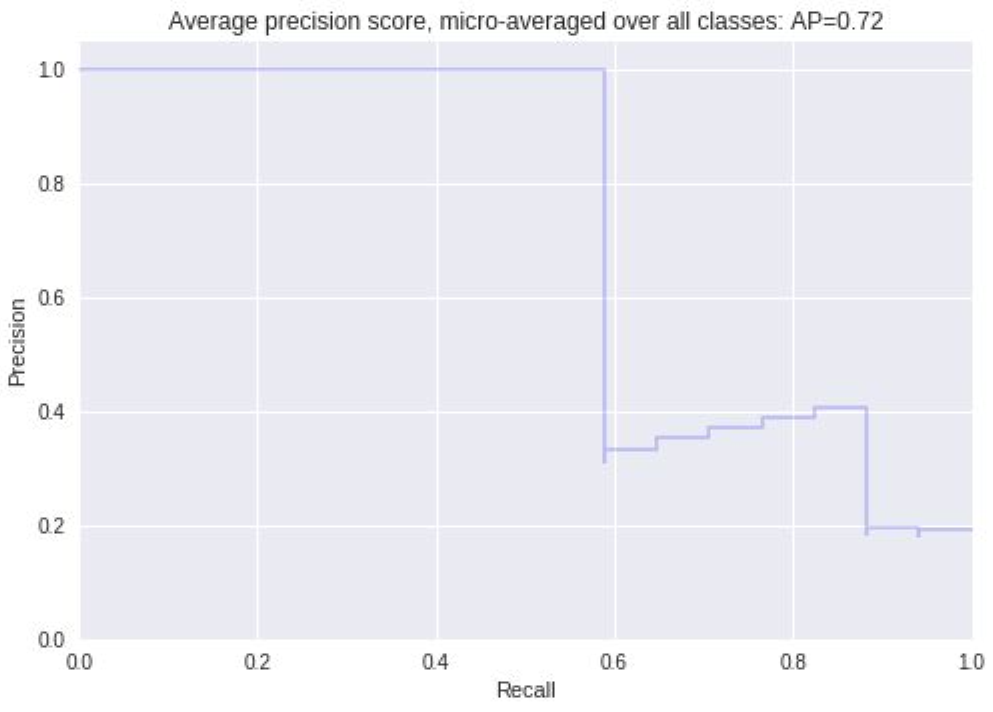
-Linear Kernel of SVC :



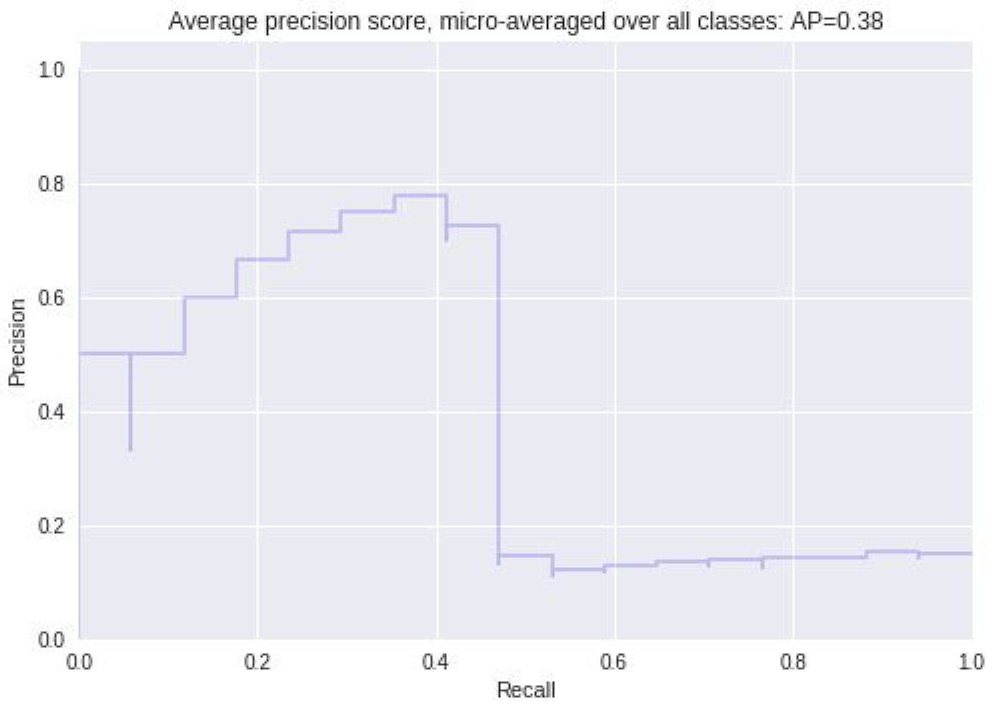
- RBF kernel of SVC :



- Polynomial kernel of SVC :



- Sigmoid kernel of SVC :



Inference : Best ROC curve is given by linear kernel of SVC. This is because the data is linearly separable and the linear kernel tends to capture all the features with much less overfitting as compared to other kernels and is computationally inexpensive too.

Task d) Link to code and explanation
Link to code solution :
https://github.com/epicalyx/Celestini_2019/blob/master/Sec_B/model.ipynb

Link to README files for detailed explanation, code solution and other files:
https://github.com/epicalyx/Celestini_2019/tree/master/Sec_B