BDMH Report File

Group:- 1041
Members :-

1) Rohit Roy - 2018259
2) Shanu Verma - 2019104
3) Jasdeep Singh - 2019047

What we have tried out:

We have tried out many models for classification in this assignment. Some of the notable ones are KNNClassifier,RandomForest Classifier and Bagging Classifier.

At first, we used simple feature extraction techniques like One-Hot encoding the amino acid sequence or extracting the composition features of amino acids using BioPython. One-Hot encoding was giving poor results and BioPython was giving relatively good results. Then we thought of trying different models using some default parameters.

First we tried these models using default parameters and it is not giving us good results then we tried to tune the parameters using GridSearchCV. We tried dimensionality reduction techniques like PCA and tried to standardize and normalize our data. But we saw that techniques like PCA were only reducing the accuracy. One reason for this is that by using PCA, we were only losing the overall data given to us, hence the lesser accuracy. Still, after parameter-tuning we were not able to get good results.

As we saw that using different models and parameters is not giving much better results, we felt that we might need to change the features to achieve better results. We learned of various composition features corresponding to amino acid sequences so we used the library propy to extract dipeptide features of the amino acid sequences. We learnt that we can get dipeptide features of an amino acid sequence using the method **GetDPComp()** from the module propy.PyPro. After using the dipeptide features, we saw that our accuracy increased.

Then, we explored and found out that if we combine output from more than one classifier and predict using multiple classifiers instead of just one then it can increase the accuracy. So we used sklearn's StackingClassifier where we used RandomForest, KNNClassifier,MLPClassifier ,SVM classifier as base estimator and logistic regression as final_estimator. And also for feature extraction we used pypro. After using StackingClassifier and parameter tuning, we were able to increase accuracy to 57.12%.

Techniques we used :-

1) Random forest as a base estimator in StackingClassifier
2) SVC as a base estimator in StackingClassifier
3) KNNClassifier as another base estimator in StackingClassifier
4) MLPClassifier as another base estimator in StackingClassifier
5) LogisticRegression as the final estimator in StackingClassifier
6) StackingClassifier as an Ensemble classifier
7) Dipeptide features of amino acids from the module propy.PyPro

How to execute code ?

Before executing code we have to first install import libraries -

1) Pandas - pip install pandas
2) Numpy - pip install numpy
3) Sklearn - pip install -U scikit-learn
4) Propy - pip install propy3

Command to **execute** python command : **python code.py**

The training dataset file is **train.csv**
The testing dataset file is **test.csv**
The script/code file is **code.py**
After running the code, it will generate an output file named **Output_File.csv** which will be used for submission on kaggle.

Code with comments as explanation (comments are marked in italics and bold and are enclosed between triple quotations):-

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import StackingClassifier
from propy import PyPro
import random
import warnings
```

```
warnings.filterwarnings('ignore')
"""a command used to prevent any unnecessary warnings from displaying in
the terminal"""

dataset1 = pd.read_csv('train.csv')
dataset2 = pd.read_csv('test.csv')
"""loading the training dataset (train.csv) in dataset1 and loading the
testing dataset (test.csv) in dataset2"""

dataset1 = dataset1.sample(frac=1).reset_index(drop=True)
"""We are shuffling the training dataset. We are doing this because we
can see in the .csv file of training dataset that same output labels are
adjacently placed. This gives a possibility that the model may not be
able to train properly. To prevent that, we have the use shuffling"""

xtrain = dataset1.iloc[:,0].values
ytrain = dataset1.iloc[:,1].values
xtest = dataset2.iloc[:,1].values

"""xtrain is 1-D numpy array such that xtrain[i] is the ith amino-acid
sequence string in the training dataset.
ytrain is 1-D numpy array such that ytrain[i] is the ith output label in
the training dataset.
xtest is 1-D numpy array such that xtest[i] is the ith amino-acid
sequence string in the testing dataset."""


def convert_xdata_to_PyPro_features(x):
      new_x = np.empty((0,400), float)
      for i in range(len(x)):
            sequence = x[i]
            obj = PyPro.GetProDes(sequence)
            l = np.array(list(obj.GetDPComp().values()))
            new_x = np.append(new_x, np.array([l]), axis=0)
      return new_x

"""the function convert_xdata_to_PyPro_features() takes x numpy array as
input. It converts each amino-acid sequence in x into Dipeptide feature
array and retunrs us a new 2-D numpy array new_x
such that new_x[i] is the Dipeptide feature vector for x[i] amino-acid
sequence. We have extracted the Dipeptide feature vector using the
module PyPro from the library propy.
We first converted each amino-acid sequence to a PyPro.GetProDes()
object. Then we used the method GetDPComp().values() on that object
which returns us the Dipeptide feature vector for that sequence.
We then converted that feature vector into a numpy array list and then
appened it to new_x. We did this for all the amino-acid sequence in x
and after that, we returned new_x"""
```

```
xtrain = convert_xdata_to_PyPro_features(xtrain)
xtest = convert_xdata_to_PyPro_features(xtest)

"""Since convert_xdata_to_PyPro_features(xtrain) returns us a 2-D numpy
array of feature vectors of the amino-acid sequences in xtrain, so,
xtrain now becomes the 2-D numpy array where xtrain[i] is the dipeptide
feature vector for the ith amino-acid sequence in training dataset.
Since convert_xdata_to_PyPro_features(xtest) returns us a 2-D numpy
array of feature vectors of the amino-acid sequences in xtest, so, xtest
now becomes the 2-D numpy array where xtest[i] is the dipeptide feature
vector for the ith amino-acid sequence in testing dataset.
"""



clf1 = RandomForestClassifier(n_estimators = 900, max_depth = 55)
clf2 = MLPClassifier(learning_rate_init = 0.005, alpha = 0.003, max_iter
= 260)
clf3 = KNeighborsClassifier(n_neighbors = 3)
clf4 = SVC(gamma = 'auto', kernel = 'rbf', C = 7)
clf5 = LogisticRegression(max_iter = 1200)

""" Here we define clf1 as an RandomForest Classifier, clf2 as a MLP
classifier, clf3 as KNN classifier, clf4 as a SVC regression classifier
and clf5 as a Logistic regression classifier
We will use clf1,clf2,clf3,clf4 as base estimators and clf5 as the final
estimator in a stacking classifier. We will then use that stacking
classifier to finally predict our output."""


classifier = StackingClassifier(estimators =
[('rf',clf1),('mlp',clf2),('knn',clf3),('svc',clf4)], final_estimator =
clf5)
classifier.fit(xtrain, ytrain)
""" Here we create the stacking classifier as we described above. Then
we fit it on (xtrain,ytrain), where xtrain is the feature matrix of the
training data and ytrain is the output label of the training data"""


y_pred = classifier.predict(xtest)
"""y_pred is the output labels of the classifier on the testing data
(xtest)."""


label = y_pred
ID = []
```

```
for i in range(len(label)):
      ID.append(1000+i+1)

dic = {'ID':ID, 'Lable':label}
df = pd.DataFrame(dic)
df.to_csv("Output_File.csv",index = False)
"""Here, we converted the output labels into a dataframe df. Then we
save the dataframe df to a .csv file named Output_File.csv which gets
stored in the same working directory."""
```