Modulation Recognition

Hania Hani 4697
Aya Kandil 4699
Monique Ehab 4928

First : The Data

We have 20 snr value and 10 modulation type
This combination identifies a sample where every sample in raw data is presented using two vectors each of them has 128 elements.

The import and reading of data is as follow :

```
# Xd is a dictionnary of data:
# its format is (mod,snr) : ndarray of shape(6000,2,128)
# in other word its format is (b'8PSK',2) : ndarray
Xd = pickle.load(open("/content/drive/My Drive/Colab Notebooks/RML2016.10b.dat",'rb'), encoding='bytes')

# snrs values are : [-20, -18, -16, -14, -12, -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
# mods values are : [b'8PSK', b'AM-DSB', b'BPSK', b'CPFSK', b'GFSK', b'PAM4', b'QAM16', b'QAM64', b'QPSK', b'WBFM']
snrs, mods = map(lambda j: sorted(list(set(map(lambda x: x[j], Xd.keys())))), [1, 0])

# get data & its label
# data will be the ndarray of a certain combination of (snr,mod)
# we have 6000*200 sample, each sample have an 2*128 ndarray value
# so basically up to this point X = (1200000, 2, 128)
X = []
lbl = []
for mod in mods:
    for snr in snrs:
        X.append(Xd[(mod, snr)])
        for i in range(Xd[(mod, snr)].shape[0]):
          lbl.append((mod, snr))
X = np.vstack(X)
```

Kindly Note that we used another form of import to make it easier for the classifiers which is :

```
# get data & its label
# for simple model : data is sepearated by snr value
# meaning snr of -20 have 10*6000 sample of 2*128 shape
X = []
x_extra = []
lbl = []
lbl_extra = []
for snr in snrs:
  for mod in mods :
    x_extra.append(Xd[(mod,snr)])
    for i in range(Xd[(mod,snr)].shape[0]):
      lbl_extra.append(mod)
  X.append(np.vstack(x_extra[:]))
  x_extra.clear()
  lbl.append(lbl_extra[:])
  lbl_extra.clear()

X= np.array(X)
lbl = np.vstack(lbl)
del(x_extra)
del(lbl_extra)
gc.collect()
```

Second : Feature Space

```
# features
features = {}
# Raw Time Feature
features['raw'] = X[:, 0], X[:, 1]
# First derivative in time
features['derivative'] = normalize(np.gradient(X[:, 0], axis=1)), normalize(np.gradient(X[:, 1], axis=1))
# Integral in time
features['integral'] = normalize(np.cumsum(X[:, 0], axis=1)), normalize(np.cumsum(X[:, 1], axis=1))
# All Together Feature Space
def extract_features(*arguments):
    desired = ()
    for arg in arguments:
        desired = desired + features[arg]
    return np.stack(desired, axis=1)
```

As required this help us to either operate on the data using its raw format or a combination of the features.
In case of the raw only or derivative only or integral only the sample shape is 2*128
If we add more combination for example all the sample shape is 6*128

Third : Supervised Learning

```
# split data 0.7 train 0.3 test randomly
training_data, test_data, training_labels_snrs, test_labels_snr =
train_test_split(X, lbl, test_size = 0.3, random_state = 42, stratify = lbl)

# here labels are : (b'AM-DSB', -2)
# we don't want this we want first part only so we take b'AM_DSB' only:
training_labels = [label[0] for label in training_labels_snrs]
test_labels = [label[0] for label in test_labels_snr]

# still we don't get label as string byte so transform into vector of
# 9 zeros and a single 1 indicating type [xxxx1xxx]
label_binarizer = LabelBinarizer()
label_binarizer.fit(training_labels)
y_train = label_binarizer.transform(training_labels)
label_binarizer.fit(test_labels)
y_test = label_binarizer.transform(test_labels)

# not only we want to know the type but also the snr value
snr_train = [label[1] for label in training_labels_snrs]
snr_test = [label[1] for label in test_labels_snr]
```

Here we split the data and prepare the label in the format we want

Baseline Classifiers :

Here for each snr value we fit & predict the model after splitting and doing some data preparation, as well as we keep three arrays to be able to plot the snr vs accuracy plot as required. We as well plot for each model the confusion matrix of each snr .

```
# for each snr :
acc_log = []
acc_tree = []
acc_forest = []
for i in range(0,20):
  # splitting data
  training_data, test_data, training_labels, test_labels =
  train_test_split(X[i], lbl[i], test_size = 0.3, random_state = 42)

  # Data preparation to fit model
  numberOfSample, nx, ny = training_data.shape
  x_train = training_data.reshape((numberOfSample, nx*ny))
  numberOfSample, nx, ny = test_data.shape
  x_test = test_data.reshape((numberOfSample, nx*ny))
  label_encoder = LabelEncoder()
  y_train = label_encoder.fit_transform(training_labels)
  y_test = label_encoder.fit_transform(test_labels)
```

Logistic Regression

```
# Logistic Regression
print("started logistic for snr of", snrs[i])
logistic_model = LogisticRegression(multi_class= 'multinomial',
                                    solver='newton-cg', warm_start= True,
                                    max_iter= 2000, n_jobs= 5)
logistic_model.fit(x_train,y_train)
y_pred = logistic_model.predict(x_test)
ac = metrics.accuracy_score(y_test,y_pred)
acc_log.append(ac)
cm = metrics.confusion_matrix(y_test,y_pred)
cm_df = pd.DataFrame(cm, index = mods, columns= mods)
plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df)
plt.title(f'lg_cm_snr={snrs[i]}\nAccuracy:{ac}')
plt.show()
gc.collect()
```

Decision Tree

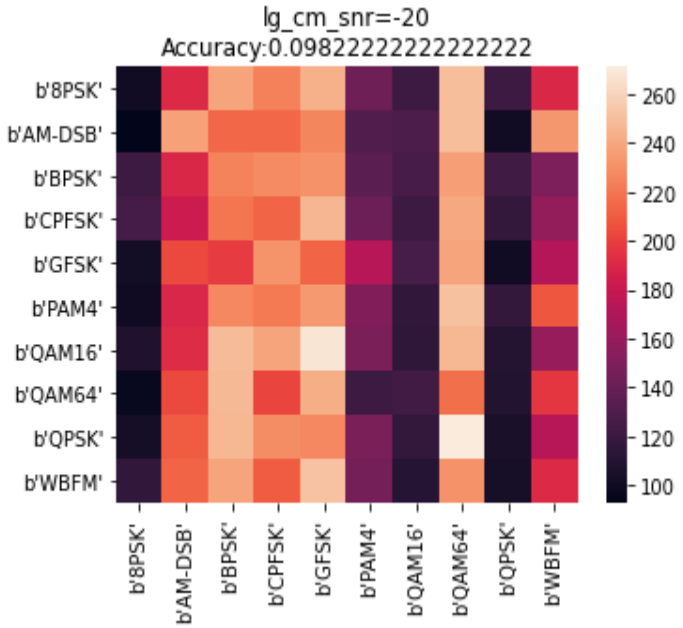
```
# Decision Tree
print("started tree for snr of", snrs[i])
tree_model = DecisionTreeClassifier(max_depth=70, min_samples_leaf=50)
tree_model.fit(x_train,y_train)
y_pred = tree_model.predict(x_test)
ac = metrics.accuracy_score(y_test, y_pred)
acc_tree.append(ac)
cm = metrics.confusion_matrix(y_test,y_pred)
cm_df = pd.DataFrame(cm, index = mods, columns= mods)
plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df)
plt.title(f'dt_cm_snr={snrs[i]}\nAccuracy:{ac}')
plt.show()
gc.collect()
```

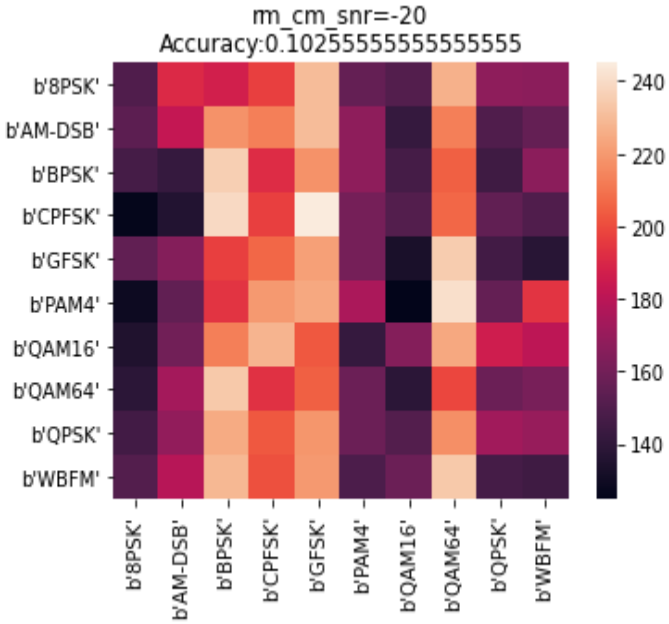
Random Forest

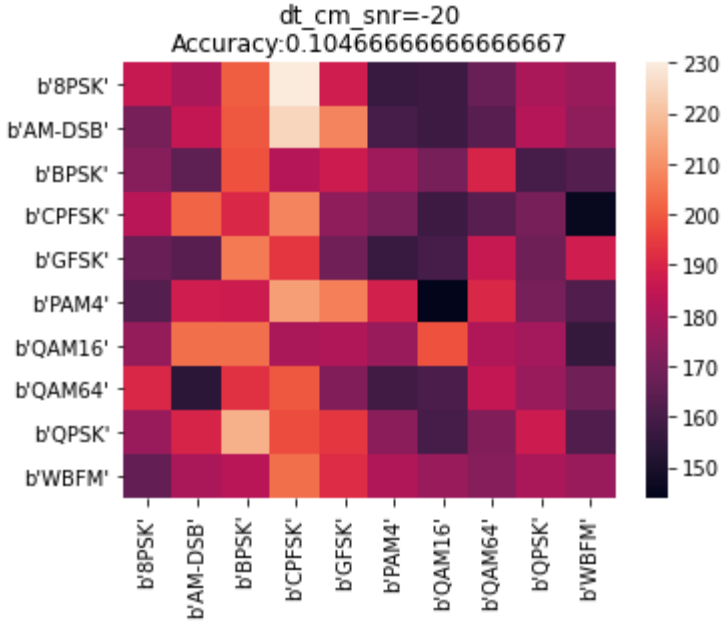
```
# Random Forest
print("started forest for snr of", snrs[i])
forest_model = RandomForestClassifier(min_samples_leaf=50, n_jobs=5, warm_start=True)
forest_model.fit(x_train,y_train)
y_pred =  forest_model.predict(x_test)
ac = metrics.accuracy_score(y_test,y_pred)
acc_forest.append(ac)
cm = metrics.confusion_matrix(y_test,y_pred)
cm_df = pd.DataFrame(cm, index = mods, columns= mods)
plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df)
plt.title(f'rm_cm_snr={snrs[i]}\nAccuracy:{ac}')
plt.show()
gc.collect()
```

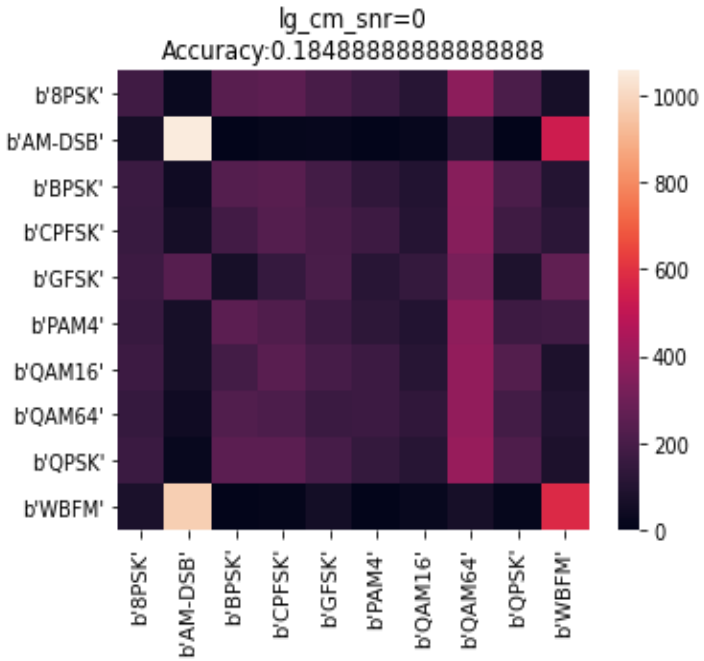
Outputs of “RAW” data :

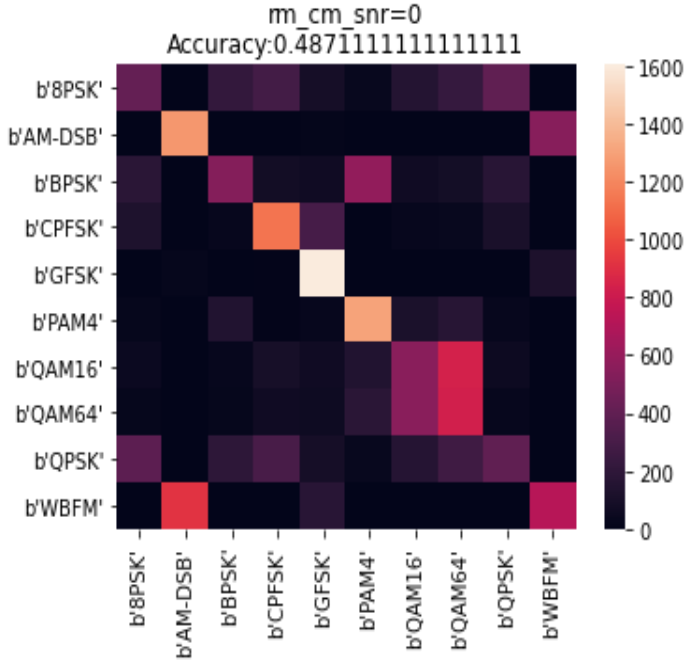
Since there is so many outputs regarding those three classifiers,
this is just a clarification of snr values [-20,0,18]
(lg=logistic regression, rm=random forest, dt=decision tree)

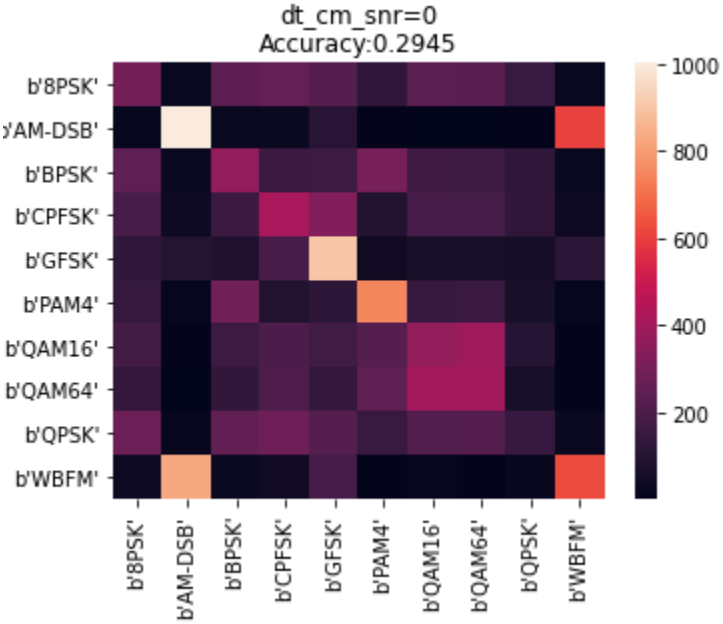


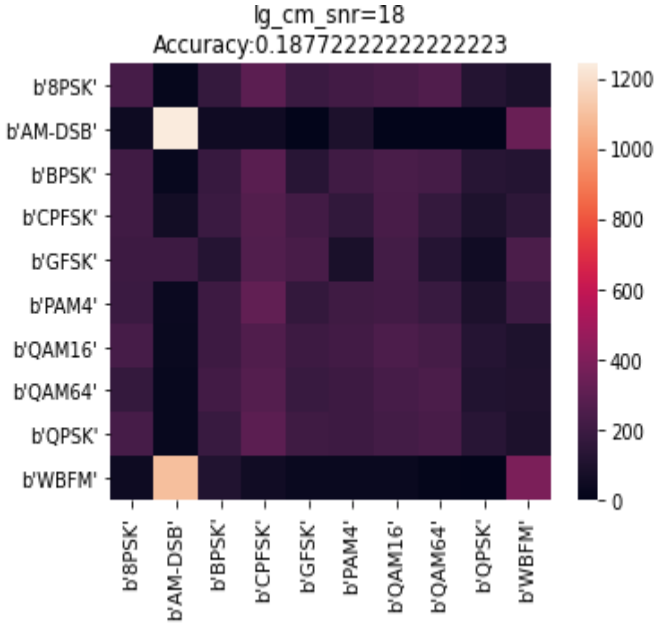


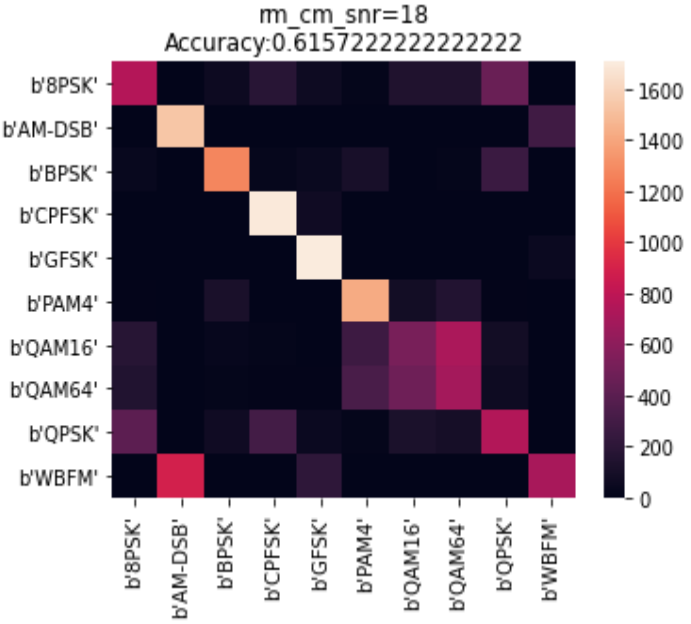


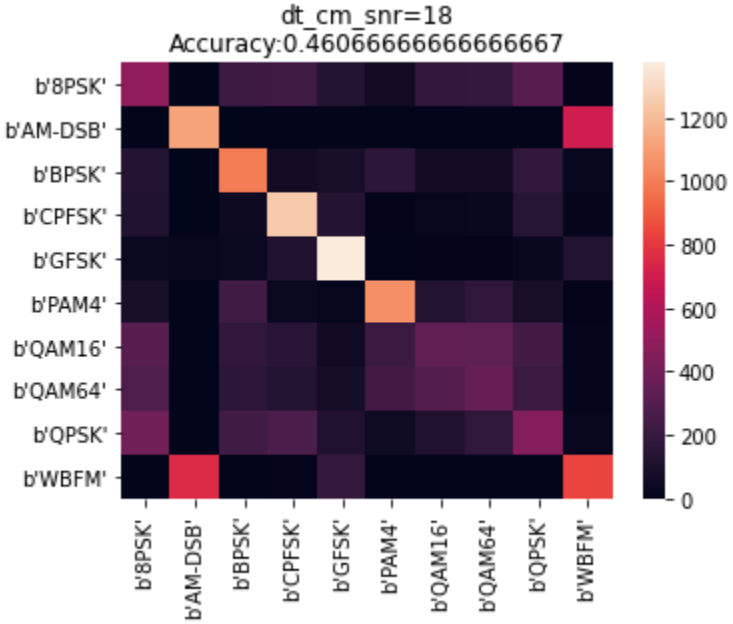

lg_cm_snr=0
Accuracy:0.18488888888888888
b'8PSK'
b'AM-DSB'
b'BPSK'
b'CPFSK'
b'GFSK'
b'PAM4'
b'QAM16'
b'QAM64'
b'QPSK'
b'WBFM'
1000
800
600
400
200
0


rm_cm_snr=0
Accuracy:0.48711111111111111
b'8PSK'
b'AM-DSB'
b'BPSK'
b'CPFSK'
b'GFSK'
b'PAM4'
b'QAM16'
b'QAM64'
b'QPSK'
b'WBFM'
1600
1400
1200
1000
800
600
400
200
0


dt_cm_snr=0
Accuracy:0.2945
b'8PSK'
b'AM-DSB'
b'BPSK'
b'CPFSK'
b'GFSK'
b'PAM4'
b'QAM16'
b'QAM64'
b'QPSK'
b'WBFM'
1000
800
600
400
200


lg_cm_snr=18
Accuracy:0.18772222222222223
b'8PSK'
b'AM-DSB'
b'BPSK'
b'CPFSK'
b'GFSK'
b'PAM4'
b'QAM16'
b'QAM64'
b'QPSK'
b'WBFM'
1200
1000
800
600
400
200
0


rm_cm_snr=18
Accuracy:0.6157222222222222
b'8PSK'
b'AM-DSB'
b'BPSK'
b'CPFSK'
b'GFSK'
b'PAM4'
b'QAM16'
b'QAM64'
b'QPSK'
b'WBFM'
1600
1400
1200
1000
800
600
400
200
0


dt_cm_snr=18
Accuracy:0.46066666666666667
b'8PSK'
b'AM-DSB'
b'BPSK'
b'CPFSK'
b'GFSK'
b'PAM4'
b'QAM16'
b'QAM64'
b'QPSK'
b'WBFM'
1200
1000
800
600
400
200
0

Rest of output of RAW data of three classifiers

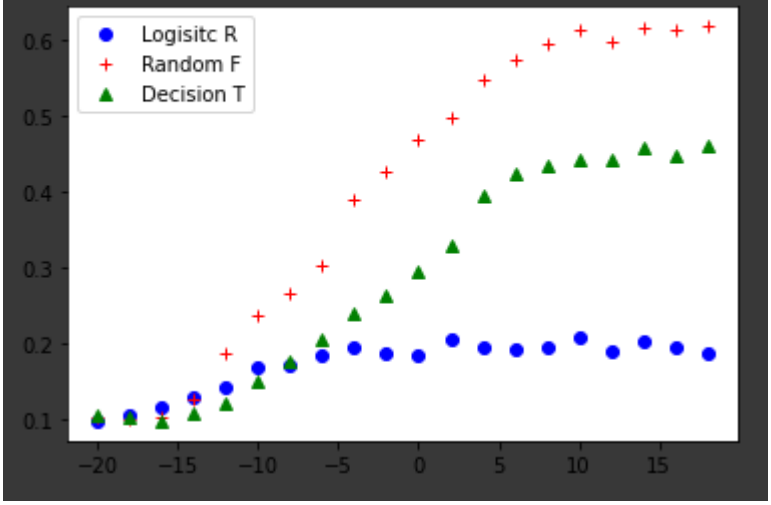
Code :

```
# calculating average accuracy :
print (f'average of logistic={sum(acc_log)/len(acc_log)}')
print (f'average of forest={sum(acc_forest)/len(acc_forest)}')
print (f'average of tree={sum(acc_tree)/len(acc_tree)}')

# plotting snr vs accuracy of each classifier
plt.figure()
fig1, = plt.plot(snrs,acc_log,'bo')
fig2, = plt.plot(snrs,acc_forest,'r+')
fig3, = plt.plot(snrs,acc_tree,'g^')
plt.legend((fig1, fig2, fig3), ('Logisitc R', 'Random F', 'Decision T'))
plt.show()
```

Output :

```
average of logistic=0.17245555555555556
average of forest=0.3991305555555556
average of tree=0.284825
```



After some changes, improved average accuracy when :

1) random forest : reducing min_samples_leaf from 50 to 20
2) decision tree : incrementing max_depth to 200 and reducing min_samples_leaf to 20

```
average of logistic=0.17245555555555556
average of forest=0.4173555555555556
average of tree=0.29033333333333333
```