Data Stream Mining Project

Emre Can Şen

21902516

Introduction

The project was done using Python. For the first part of the project, two synthetic datasets with drift were generated with AGRAWAL and SEA methods respectively, and the two real datasets were imported to the work-space. Then the classification models that were given in the report were implemented using Python libraries. Finally, the HoeffdingTreeClassifier ensemble method was implemented from scratch and accuracy of all the classifiers were reported.

Interleaved Test-Then-Train method was used for training and for the accuracy evaluation of the classifiers for each dataset. Prequential accuracy plot with sliding windows and overall accuracy was reported for all the classifiers and datasets. For the AGRAWALDataset, sliding windows prequential plots as well as total accuracy vs. total number of samples prequential plots were given. This was done for one dataset because this step was not required for the project but provided good insight on the overall performance of the classifiers.

Results

ARF- AGRAWALDataset

Prequential Evaluation

```
#################### [100%] [608.47s]
ARF - Accuracy : 0.9890
```

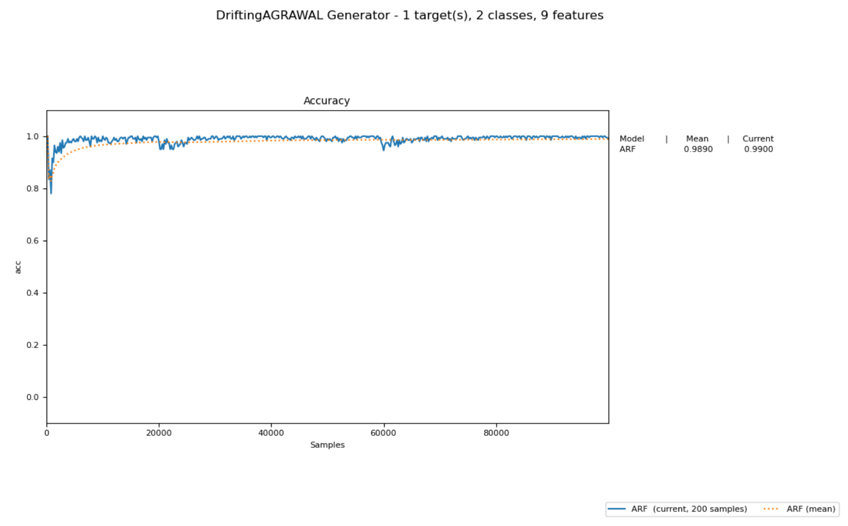

Figure 1- Prequential Accuracy Plot(ARF-ARGWAL)

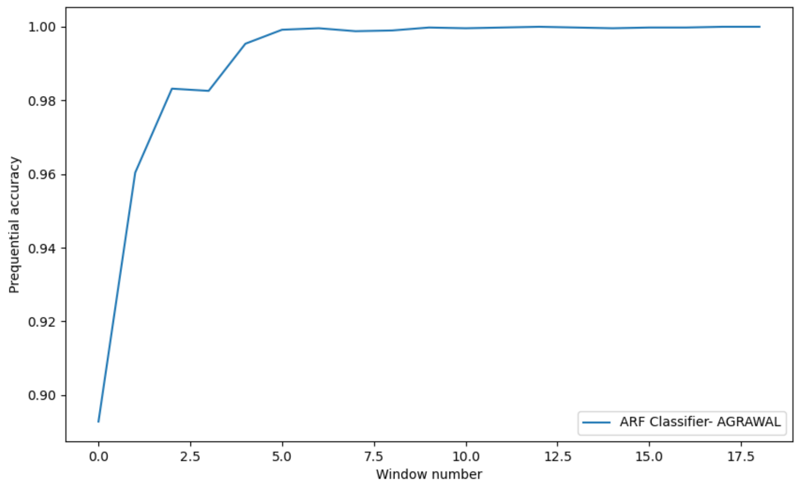

Figure 2- Sliding Window Prequential Accuracy Plot(ARF-ARGWAL)

Overall accuracy of ARF Classifier- AGRAWAL: 0.9905

SAM-KNN- AGRAWALDataset

Prequential Evaluation

```
#################### [100%] [71.11s]
SAM_kNN - Accuracy     : 0.6632
```

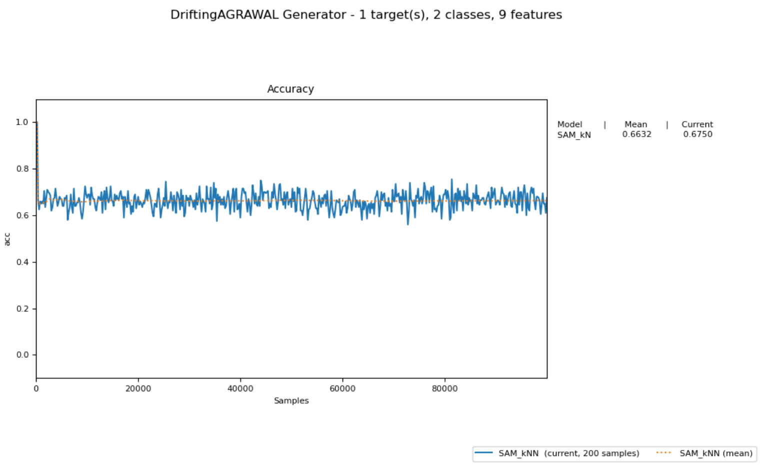

Figure 3- Prequential Accuracy Plot(SAM-KNN-ARGWAL)

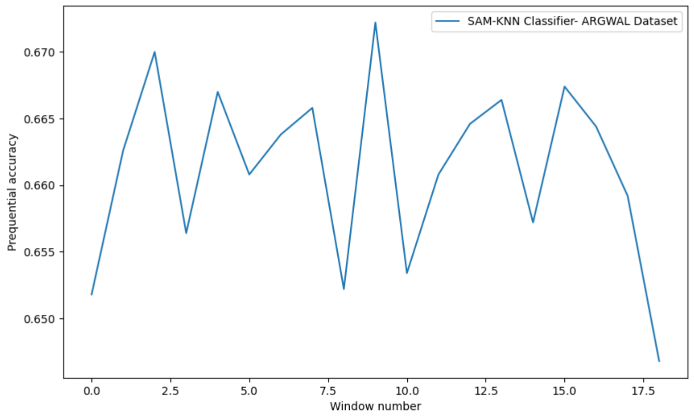

Figure 4- Sliding Window Prequential Accuracy Plot(SAM-KNN-ARGWAL)

Overall accuracy of SAM-KNN Classifier- ARGWAL Dataset: 0.6614

<u>SRP- AGRAWALDataset</u>

Prequential Evaluation

########------------ [40%] [3037.54s]

SRP - Accuracy   : 1.0000

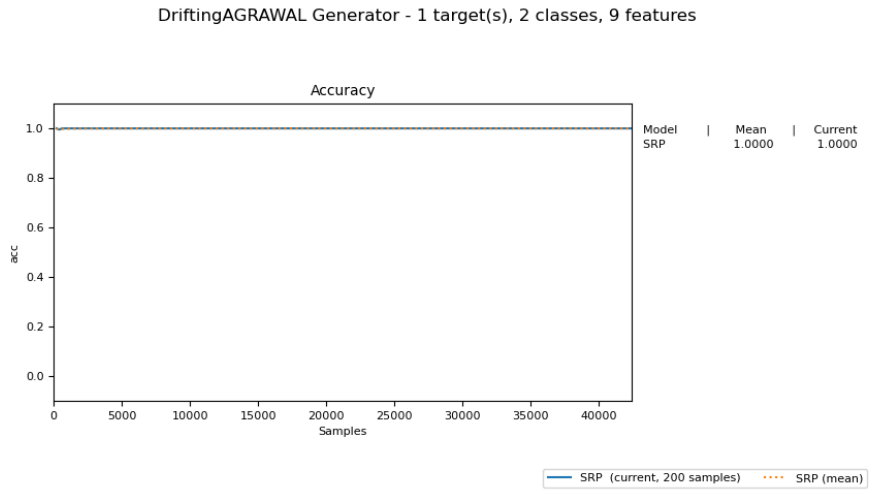

Figure 5- Prequential Accuracy Plot(SRP-ARGWAL)

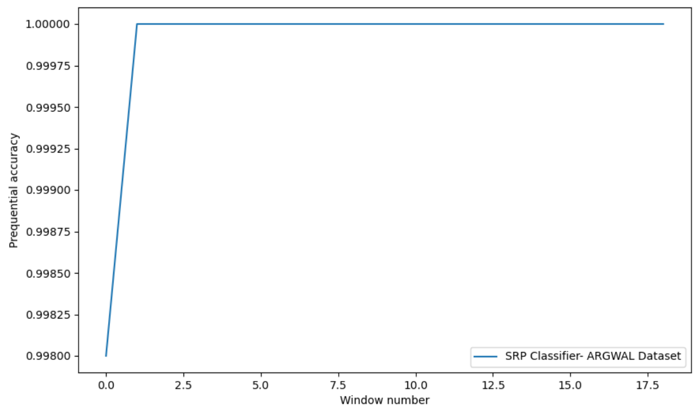

Figure 6- Sliding Window Prequential Accuracy Plot(SRP-ARGWAL)

Overall accuracy of SRP Classifier- ARGWAL Dataset: 0.9999

For ARGWAL dataset, SRP method was bugged for me. I tried decreasing the number of classifiers but kept getting the accuracy as 1.

<u>DWM- AGRAWALDataset</u>

Prequential Evaluation

#################### [100%] [103.72s]

DWM - Accuracy   : 0.8771

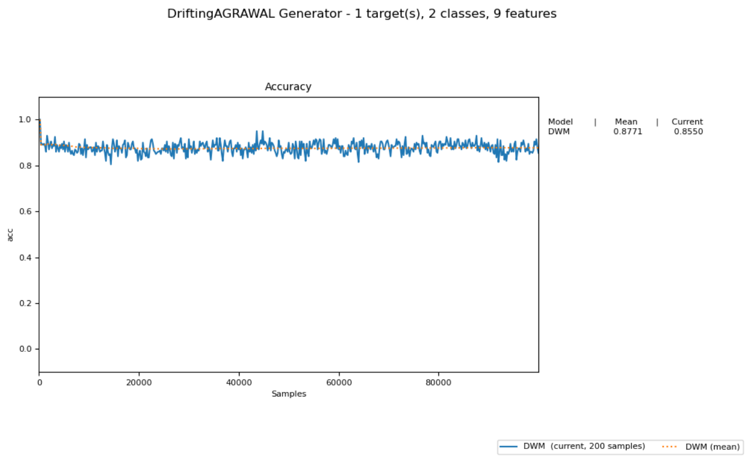

Figure 7- Prequential Accuracy Plot(DWM-ARGWAL)

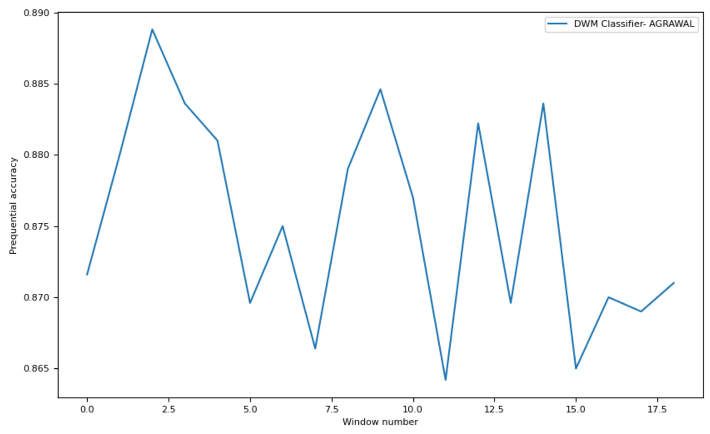

Figure 8- Sliding Window Prequential Accuracy Plot(DWM-ARGWAL)

Overall accuracy of DWM Classifier- AGRAWAL: 0.8754

<u>Ensemble- AGRAWALDataset</u>

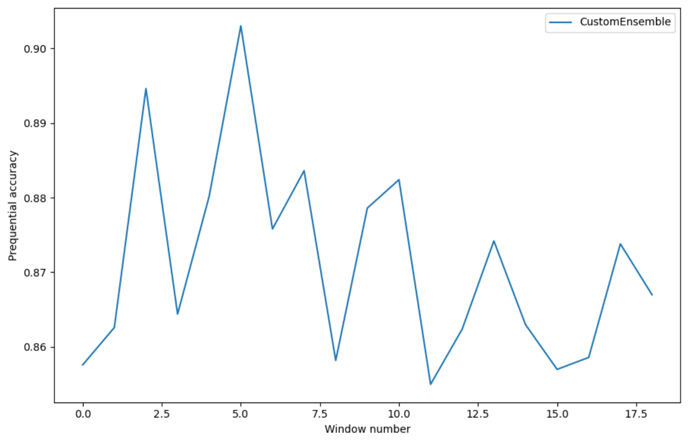

Figure 9- Sliding Window Prequential Accuracy Plot(Ensemble-ARGWAL)

Overall accuracy of CustomEnsemble: 0.8706

<u>ARF- SEADataset</u>

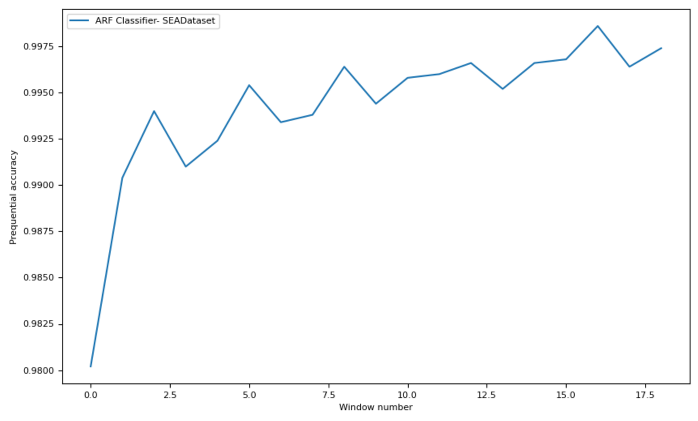

Figure 10- Sliding Window Prequential Accuracy Plot(ARF-SEA)

Overall accuracy of ARF Classifier- SEADataset: 0.9943

<u>SAM-KNN- SEADataset</u>

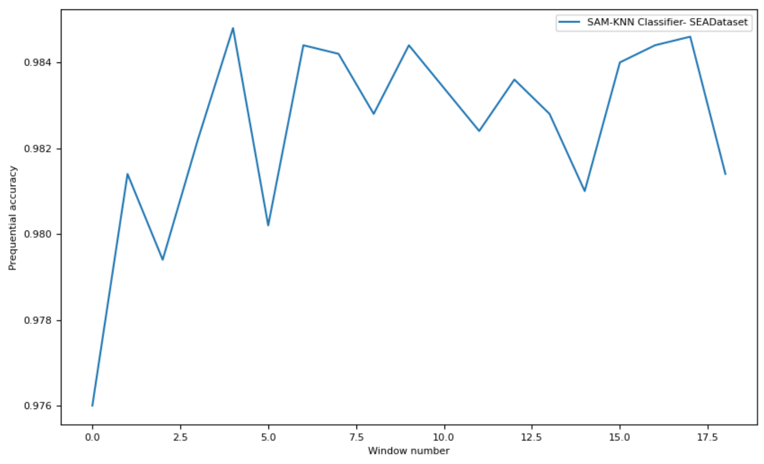

Figure 11- Sliding Window Prequential Accuracy Plot(SAM-KNN-SEA)

Overall accuracy of SAM-KNN Classifier- SEADataset: 0.9827

<u>SRP- SEADataset</u>

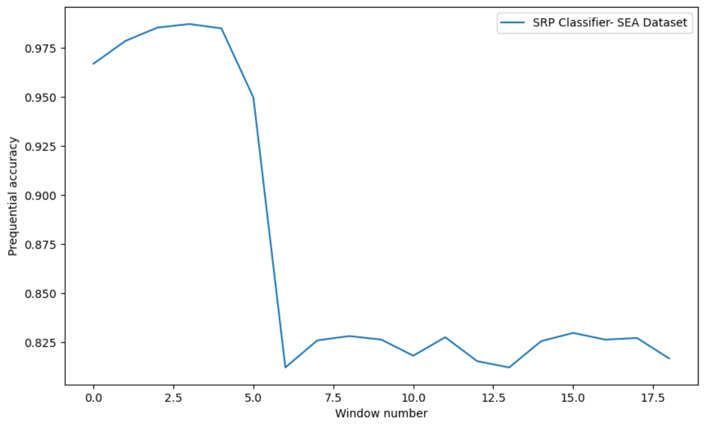

Figure 12- Sliding Window Prequential Accuracy Plot(SRP-SEA)

Overall accuracy of SRP Classifier- SEA Dataset: 0.8683

<u>DWM- SEADataset</u>

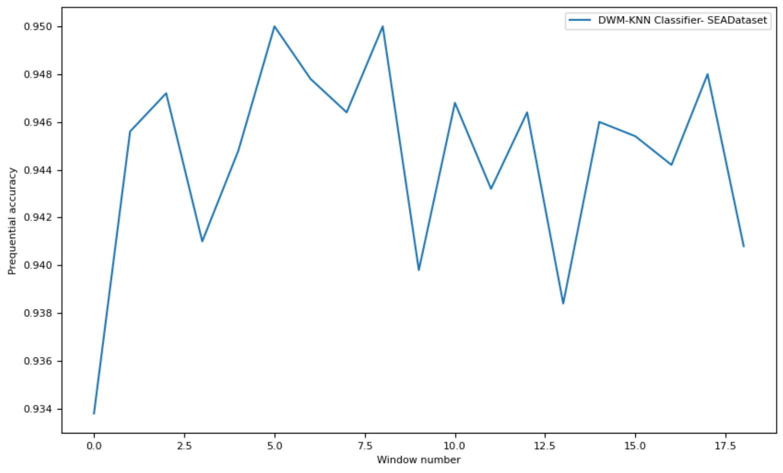

Figure 13- Sliding Window Prequential Accuracy Plot(DWM-SEA)

Overall accuracy of DWM-KNN Classifier- SEADataset: 0.9446

<u>Ensemble- SEADataset</u>

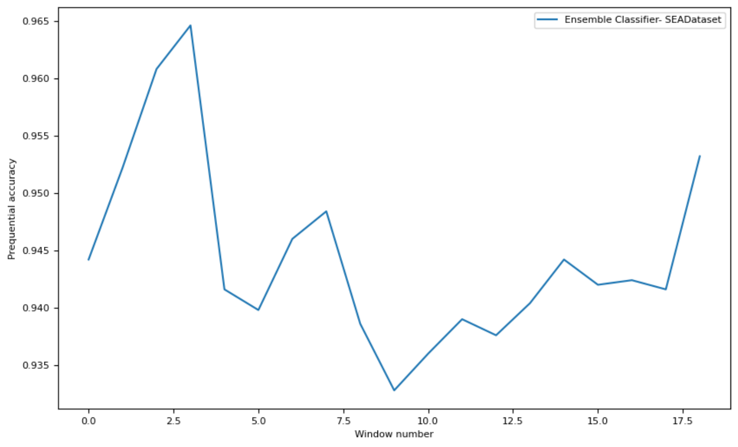

Figure 14- Sliding Window Prequential Accuracy Plot(Ensemble-SEA)

Overall accuracy of Ensemble Classifier- SEADataset: 0.9445

<u>ARF- Spam Dataset</u>

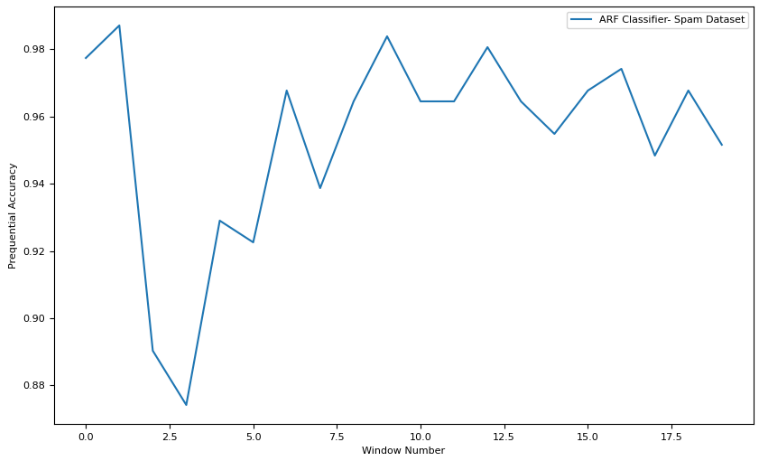

Figure 15- Sliding Window Prequential Accuracy Plot(ARF-Spam)

Overall accuracy of ARF Classifier- Spam Dataset: 0.9538

<u>SAM-KNN- Spam Dataset</u>

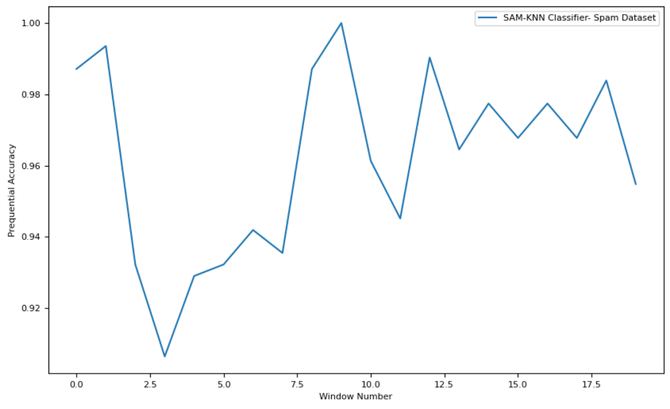

Figure 16- Sliding Window Prequential Accuracy Plot(SAM-KNN-Spam)

Overall accuracy of SAM-KNN Classifier- Spam Dataset: 0.9619

<u>SRP- Spam Dataset</u>

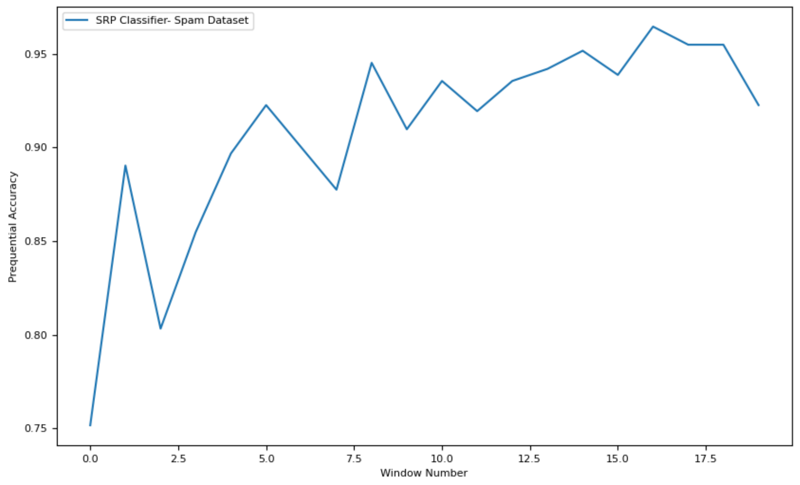

Figure 17- Sliding Window Prequential Accuracy Plot(SRP-Spam)

Overall accuracy of SRP Classifier- Spam Dataset: 0.9087

<u>DWM- Spam Dataset</u>

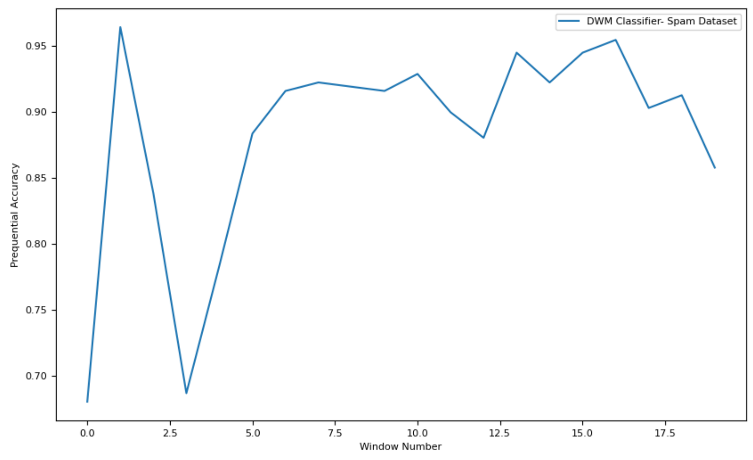

Figure 18- Sliding Window Prequential Accuracy Plot(DWM-Spam)

Overall accuracy of DWM Classifier- Spam Dataset: 0.8835

<u>Ensemble- Spam Dataset</u>

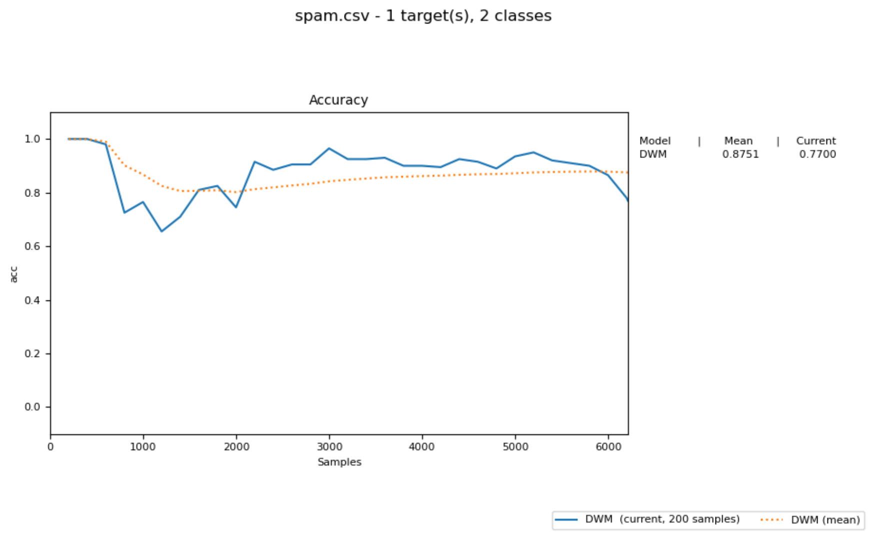

Figure 19- Prequential Accuracy Plot(Ensemble-Spam)

Accidentally wrote DWM on plot but ensemble.

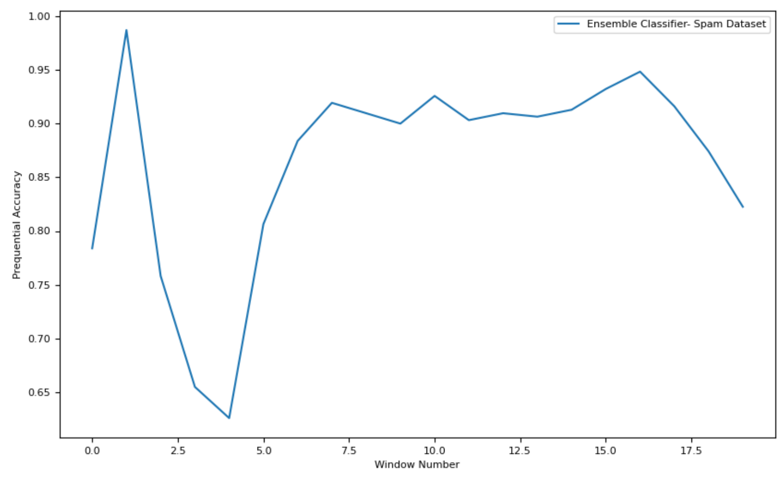

Figure 20- Sliding Window Prequential Accuracy Plot(Ensemble-Spam)

Overall accuracy of Ensemble Classifier- Spam Dataset: 0.8640

ARF- Electricity Dataset

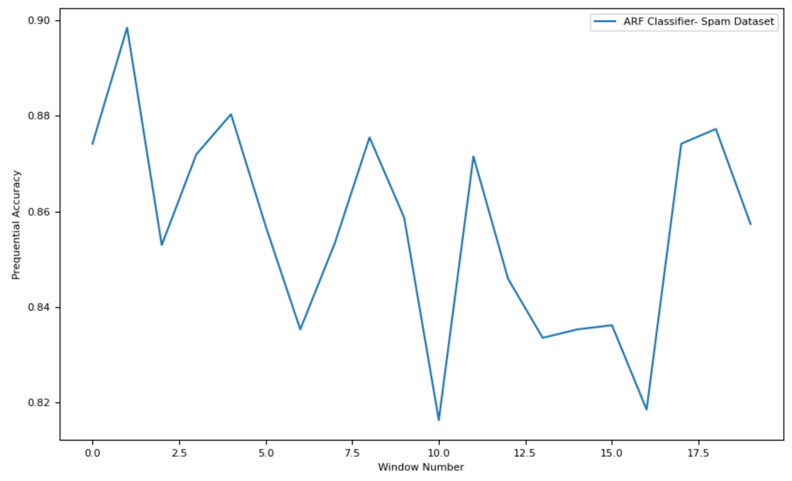

Figure 21- Sliding Window Prequential Accuracy Plot(ARF-Electricity)

Overall accuracy of ARF Classifier- Spam Dataset: 0.8562

SAM-KNN- Electricity Dataset

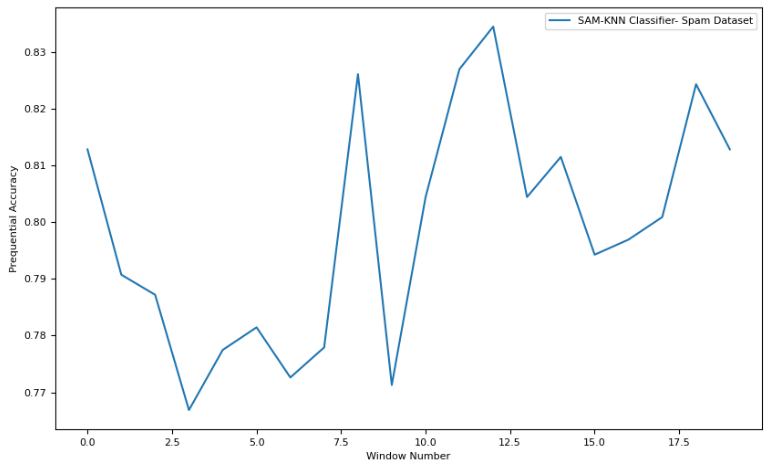

Figure 22- Sliding Window Prequential Accuracy Plot(SAM-KNN-Electricity)

Overall accuracy of SAM-KNN Classifier- Spam Dataset: 0.7988

<u>SRP- Electricity Dataset</u>

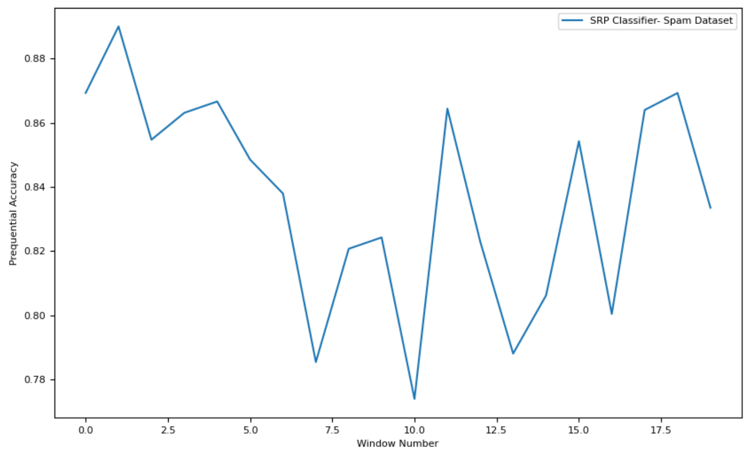

Figure 23- Sliding Window Prequential Accuracy Plot(SRP-Electricity)

Overall accuracy of SRP Classifier- Spam Dataset: 0.8369

<u>DWM- Electricity Dataset</u>

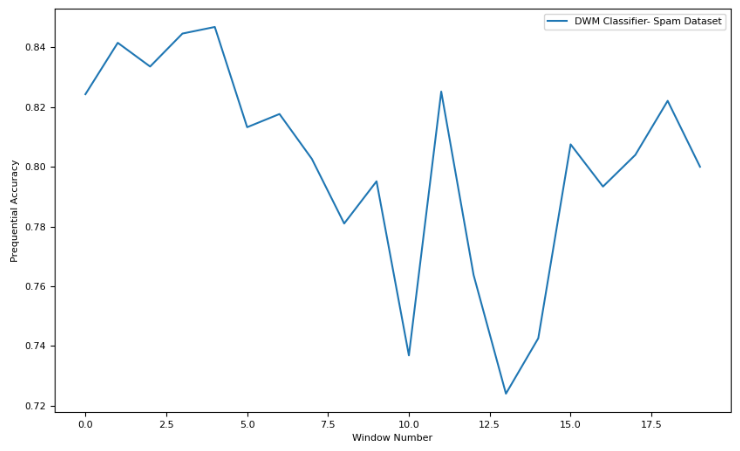

Figure 24- Sliding Window Prequential Accuracy Plot(DWM-Electricity)

Overall accuracy of DWM Classifier- Spam Dataset: 0.8010

<u>Ensemble- Electricity Dataset</u>

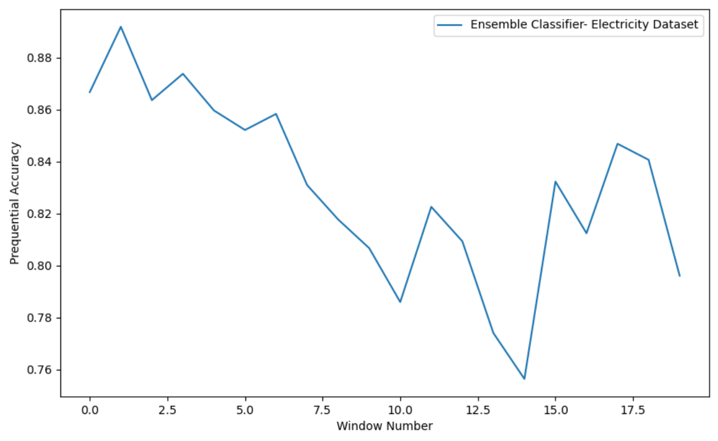

Figure 25- Sliding Window Prequential Accuracy Plot(Ensemble-Electricity)

Overall accuracy of Ensemble Classifier- Electricity Dataset: 0.8298

Discussion

For the electricity dataset ensemble performed better than DWM and SRP but the ensemble training time was much greater than those two methods. On spam dataset, ensemble classifier performed the worst across the classifiers. On synthetic datasets, ARF classifier performed the best, but this is probably due to the fact that ARF classifier training took the most time, reflecting the model’s complexity. For better results, model’s complexity can be adjusted to be more similar. Not only ARF but other model’s accuracy was also high, this might be due to generated synthetic datasets having less drift compared to real datasets. So ensemble models did not perform particularly well in this case compared to other classifiers.

To enhance the robustness of the ensemble, we can include the utilization of advanced drift detection mechanisms, the incorporation of a greater variety of base learners into the ensemble, and the exploration of ensemble learning strategies such as stacking or boosting. By implementing these enhancements, we can potentially improve the ensemble’s performance.

The declines in the prequential accuracy plot signify intervals during which the models encounter a decline in their predictive accuracy. This decline can be attributed to concept drift in the data, which means the alteration of the underlying data distribution and resulting in a deterioration of the model's performance. These accuracy drops emphasize the necessity of effectively adapting the models to address concept drift, which may involve updating the models.

For this project, the focus was on implementing and assessing classification models made for data streams with concept drift. Employing the Interleaved Test-Then-Train methodology, both the overall accuracy and prequential accuracy of these models were computed. By comparing the ensemble model's performance against state-of-the-art approaches, potential areas for enhancement were underlined. The prequential accuracy plot proved invaluable in discerning the model's performance trends, revealing accuracy declines during concept drift phases. Ultimately, this assignment provided practical experience in effectively managing concept drift in data streams and evaluating classification model performance under such circumstances.

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 26 20:24:48 2023

@author: Emre
"""
```

```
import pandas as pd
import numpy as np
import random
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt

def k_means(dataset, K, iter_cnt, outliers=True):
    #Initialize dataset size
    m = dataset.shape[0]
    n = dataset.shape[1]
    centroid_array = np.array([]).reshape(n, 0)
    for i in range(K):
        centroid_array = np.c_[centroid_array, dataset[random.randint(0, m-1)]]
    #euclidian distances array
    euclid_array = np.array([]).reshape(m, 0)
    #centroid euclidian distances
    for i in range(K):
        dist = np.sum((dataset-centroid_array[:, i])**2, axis=1)
        euclid_array = np.c_[euclid_array, dist]
    #minimum distance
    minimum = np.argmin(euclid_array, axis=1)+1
    #Centoroid processes
    #Storing the clusters
    centoroid = {}
```

```
for k in range(K):
    centoroid[k+1] = np.array([]).reshape(2, 0)
#Cluster the points
for k in range(m):
    centoroid[minimum[k]] = np.c_[centoroid[minimum[k]], dataset[k]]
for k in range(K):
    centoroid[k+1] = centoroid[k+1].T
#Compute the mean
for k in range(K):
    centroid_array[:, k] = np.mean(centoroid[k+1], axis=0)
#Store final positions of centroid and group the points
result = {}
for i in range(iter_cnt):
    euclid_array = np.array([]).reshape(m, 0)
    for k in range(K):
        dist = np.sum((dataset-centroid_array[:, k])**2, axis=1)
        euclid_array = np.c_[euclid_array, dist]
    C = np.argmin(euclid_array, axis=1)+1
    centoroid = {}
    for k in range(K):
        centoroid[k+1] = np.array([]).reshape(2, 0)
    for k in range(m):
        centoroid[C[k]] = np.c_[centoroid[C[k]], dataset[k]]
    for k in range(K):
        centoroid[k+1] = centoroid[k+1].T
    for k in range(K):
        centroid_array[:, k] = np.mean(centoroid[k+1], axis=0)
    result = centoroid
#Plot the graph
```

```
    for i in range(K):
        plt.scatter(result[i+1][:, 0], result[i+1][:, 1],
                    label=("Cluster " + str(i+1)))
    plt.legend()
    plt.title("K Clustering for k=" + str(k+1))
    plt.show()

    return result

#######Part A#######
falldetection_dataset = pd.read_csv("falldetection_dataset.csv", header=None)

data_feat = falldetection_dataset.drop([0, 1], axis=1)
data_label = falldetection_dataset[1]

train_data_x= data_feat.values
train_data_y = data_label.values

"""Apply PCA"""
# Center the data
train_centered_x = train_data_x - np.mean(train_data_x)
# Calculate the covariance matrix
covariance_matrix = np.cov(train_centered_x, rowvar=False)
# From the covariance matrix, eigen values and eigen vectors are calculated
eigen_values, eigen_vectors = np.linalg.eigh(covariance_matrix)
# Sort the eigen values and vectors
sorted_eigen_values = np.argsort(eigen_values)[::-1]
sorted_eigen_vectors = eigen_vectors[:, sorted_eigen_values]
```

```
# Calculate the sum of the eigen values

total_sum = eigen_values.sum()

# Calculating the top PC's

n_components = 2

pca = sorted_eigen_vectors[:, 0:n_components]


# Calculate the explained variance ratio

var_exp = eigen_values[sorted_eigen_values]

var_exp_ratio = var_exp / total_sum


pca_x = np.dot(pca.T, train_centered_x.T).T

pca_x = -1 * pca_x


for idx in range(n_components):

    print(f"Principal Component {idx+1}, Variance Explained: {var_exp_ratio[idx]}")

print("Total variance: ", sum(var_exp_ratio[:n_components]))


# Comment/Uncomment for Graphs, each graph was generated with seperate runs while the other
graph codes were commented out


# Uncomment for PCA Graphs

plt.scatter(pca_x[train_data_y == 'F', 0],

            pca_x[train_data_y == 'F', 1], label="Fall")

plt.scatter(pca_x[train_data_y == 'NF', 0],

            pca_x[train_data_y == 'NF', 1], label="Non-Fall")

plt.title('PCA with 2 principal components')

plt.legend()

plt.show()
```

```
# Initialize the indeces of the outliers
outlier_max = max(pca_x[:, 1])
outlier_min = min(pca_x[:, 1])
index_max = np.where((pca_x[:, 1] == outlier_max) == True)[0][0]
index_min = np.where((pca_x[:, 1] == outlier_min) == True)[0][0]

# Remove the outliers
pca_x_no_outliers = np.delete(pca_x, [index_max, index_min], axis=0)
train_y_no_outliers = np.delete(train_data_y, [index_max, index_min], axis=0)

# Uncomment for PCA Graphs
plt.scatter(pca_x_no_outliers[train_y_no_outliers == 'F', 0],
        pca_x_no_outliers[train_y_no_outliers == 'F', 1], label="Fall")
plt.scatter(pca_x_no_outliers[train_y_no_outliers == 'NF', 0],
        pca_x_no_outliers[train_y_no_outliers == 'NF', 1], label="Non-Fall")
plt.title('PCA without outliers and with 2 principal components')
plt.legend()
plt.show()


# Uncomment for K Clustering Graphs
k_means2 = k_means(pca_x, 2, 150, False) # with outliers
k_means3 = k_means(pca_x_no_outliers, 3, 150, False)
k_means4 = k_means(pca_x_no_outliers, 4, 150, False)
k_means6 = k_means(pca_x_no_outliers, 6, 150, False)
k_means8 = k_means(pca_x_no_outliers, 8, 150, False)
k_means10 = k_means(pca_x_no_outliers, 10, 150, False)

k_means2_outliers = k_means(pca_x_no_outliers, 2, 150, False)
```

```
## Consistency Code
kmeans = KMeans(n_clusters = 2)
kmeans.fit(pca_x) #Fitting the data with K-means
k_prediction = kmeans.labels_ #Predict the labels
## Find the accuracy/consistency score
feature_label = [1 if prediction_label == "NF" else 0 for prediction_label in train_data_y]
acc1 = accuracy_score(feature_label, k_prediction)
acc2 = accuracy_score(feature_label, 1 - k_prediction)

if (acc1 >= acc2):
    print("Accuracy of 2-means = ", acc1)
else:
    print("Accuracy of 2-means = ", acc2)

#######Part B#######
train_x, test_x, train_y, test_y = train_test_split(train_data_x, train_data_y, test_size=0.30)
valid_x, test_x, valid_y, test_y = train_test_split(test_x, test_y, test_size=0.50)

#Values of parameters for SVM
reg_parameter = [1e-3, 1e-2, 1e-1, 1e0, 1e1]
degree = [1, 2, 3, 4]
kernel = ["linear", "poly", "sigmoid", "rbf"]

svm_outputs = []
for reg in reg_parameter:
    for k in kernel:
        #Linear kernel
        if k == "linear":
```

```
    svm_model = SVC( C = reg, kernel = k, max_iter = 1000 )
    svm_model.fit(train_x, train_y)
    prediction = svm_model.predict(valid_x)
    predict_test=svm_model.predict(test_x)
    val_acc = metrics.accuracy_score(valid_y, prediction)
    acc = metrics.accuracy_score(test_y, predict_test)
    #Write Data to a .csv file
    data = {'Regularization Parameter': reg,
            'Type': k,
            'Degree of Polynomial': "",
            'Test Accuracy':acc,
            'Validation Accuracy': val_acc}
    svm_outputs.append(data)
    print(data)
#Poly kernel
elif k == "poly":
    #Check poly in multiple degrees
    for d in degree:
        svm_model = SVC( C = reg, kernel=k, degree=d, max_iter=10000 )
        svm_model.fit(train_x, train_y)
        prediction = svm_model.predict(valid_x)
        predict_test=svm_model.predict(test_x)
        val_acc = metrics.accuracy_score(valid_y, prediction)
        acc = metrics.accuracy_score(test_y, predict_test)
        data = {'Regularization Parameter': reg,
                'Type': k,
                'Degree of Polynomial': degree,
                'Test Accuracy':acc,
                'Validation Accuracy': val_acc}
            svm_outputs.append(data)
            print(data)
        else:
            #Other kernels
            svm_model = SVC( C = reg, kernel = k, max_iter = 10000 )
            svm_model.fit(train_x, train_y)
            prediction = svm_model.predict(valid_x)
            predict_test=svm_model.predict(test_x)
            val_acc = metrics.accuracy_score(valid_y, prediction)
            acc = metrics.accuracy_score(test_y, predict_test)
            data = {'Regularization Parameter': reg,
                    'Type': k,
                    'Degree of Polynomial': "",
                    'Test Accuracy':acc,
                    'Validation Accuracy': val_acc}
            svm_outputs.append(data)
            print(data)

#Sort the list of results according to validation accuracy
svm_sorted = sorted( svm_outputs, key = lambda x: x['Validation Accuracy'], reverse=True)
data_svm = pd.DataFrame(svm_sorted)

#Writing the results to the xlsx file to make easy include to the report
writer = pd.ExcelWriter('DataSVM.xlsx')
data_svm.to_excel(writer, 'Sheet1', index=False)
writer.save()
print('data_svm written to excel.')

#Showing best SVM Result
```

```
svm_best = svm_sorted[0]
reg = svm_best['Regularization Parameter']
k = svm_best['Type']
degree = svm_best['Degree of Polynomial']
val_acc = svm_best['Validation Accuracy']

print("Best SVM model:")
print("Regularization Parameter:", reg)
print("Degree:", degree)
print("Kernel Type:", k)
print("Validation Accuracy:", val_acc)

if degree != "":
   svm_best = SVC( C = reg, kernel = k, degree = degree, max_iter = 10000 )
   svm_best.fit(train_x, train_y)
   prediction = svm_best.predict(test_x)
   acc = metrics.accuracy_score(test_y, prediction)
   print("Accuracy of the best SVM: ", acc)

else:
   svm_best = SVC( C = reg, kernel = k, max_iter = 10000 )
   svm_best.fit(train_x, train_y)
   prediction = svm_best.predict(test_x)
   acc = metrics.accuracy_score(test_y, prediction)
   print("Accuracy of the best SVM: ", acc)

#Calculating MLP
# Parameters
reg = [1e-3, 1e-2, 1e-1, 1e0, 1e1]
```

```
hidden_layer = [(8, 8), (16, 16), (32, 32)]

activate = ["tanh", "relu","logistic"]

learning_rates = [1e-3, 1e-2, 1e-1]


mlp_outputs = []


#Iterate each combination of parameters

for layer_size in hidden_layer:


    for activation_function in activate:


        for r in reg:


            for learning in learning_rates:


                mlp_model = MLPClassifier(hidden_layer_sizes=layer_size, activation=activation_function,
alpha=r, learning_rate_init=learning, max_iter=5000)

                mlp_model.fit(train_x, train_y)

                prediction = mlp_model.predict(valid_x)

                predict_test=mlp_model.predict(test_x)

                val_acc = metrics.accuracy_score(valid_y, prediction)

                acc = metrics.accuracy_score(test_y, predict_test)


                data = {'Hidden Layer Size': layer_size,

                    'Activation Function': activation_function,

                    'Regularization Size': r,

                    'Learning Rate': learning,

                    'Test Accuracy': acc,

                    'Validation Accuracy': val_acc}
            mlp_outputs.append(data)

            print(data)


mlp_sorted = sorted(mlp_outputs, key=lambda x: x['Validation Accuracy'], reverse=True)

MLP_data = pd.DataFrame(mlp_sorted)


writer = pd.ExcelWriter('DataMLP.xlsx')

MLP_data.to_excel(writer, 'Sheet1', index=False)

writer.save()

print('MLP_data is written to Excel File successfully.')


best_mlp = mlp_sorted[18]

hidden_layer = best_mlp['Hidden Layer Size']

activation_function = best_mlp['Activation Function']

reg = best_mlp['Regularization Size']

learning_rate = best_mlp['Learning Rate']

val_acc = best_mlp['Validation Accuracy']


print("Best MLP model:")

print("Hidden Layer Size: ", hidden_layer)

print("Activation Function: ", activation_function)

print("Regularization Size:", reg)

print("Learning Rate: ", learning_rate)

print("Validation Accuracy: ", val_acc)


best_mlp = MLPClassifier( hidden_layer_sizes = hidden_layer, activation = activation_function, alpha =
reg, learning_rate_init = learning_rate, max_iter = 2000 )

best_mlp.fit(train_x, train_y)
```

```
prediction = best_mlp.predict(test_x)

acc = metrics.accuracy_score(test_y, prediction)

print("Accuracy of the best MLP: ", acc)
```