```
! pip install -U scikit-multiflow

# Email the TA if the exact code can be used and how to give reference to it.
```

```
Collecting scikit-multiflow
  Downloading scikit_multiflow-0.5.3-cp37-cp37m-manylinux2010_x86_64.whl (1.1
     |████████████████████████████████| 1.1 MB 16.9 MB/s
Requirement already satisfied: sortedcontainers>=1.5.7 in /usr/local/lib/pythc
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/
Requirement already satisfied: pandas>=0.25.3 in /usr/local/lib/python3.7/dist
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /us
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/d
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/d
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-p
Installing collected packages: scikit-multiflow
Successfully installed scikit-multiflow-0.5.3
```

```
import skmultiflow as mf
import numpy as np
import pandas as pd
from skmultiflow.data.file_stream import FileStream
#https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.dat

import matplotlib.pyplot as plt
import time
```

```
HG_a = mf.data.HyperplaneGenerator(n_features=10, n_drift_features=2, noise_perc
x_a, y_a = HG_a.next_sample(20000) #https://scikit-multiflow.readthedocs.io/en/s

HG_b = mf.data.HyperplaneGenerator(n_features=10, n_drift_features=2, noise_perc
x_b, y_b = HG_b.next_sample(20000)

HG_c = mf.data.HyperplaneGenerator(n_features=10, n_drift_features=5, noise_perc
x_c, y_c = HG_c.next_sample(20000)

HG_d = mf.data.HyperplaneGenerator(n_features=10, n_drift_features=5, noise_perc
x_d, y_d = HG_d.next_sample(20000)
```

```

# https://scikit-multiflow.readthedocs.io/en/stable/user-guide/streams-intro.htm
ds_a = pd.DataFrame(np.hstack((x_a,np.array([y_a]).T)))
ds_b = pd.DataFrame(np.hstack((x_b,np.array([y_b]).T)))
ds_c = pd.DataFrame(np.hstack((x_c,np.array([y_c]).T)))
```

```
ds_d = pd.DataFrame(np.hstack((x_d,np.array([y_d]).T)))
```

```
ds_a.to_csv("Hyperplane Dataset 10_2.csv", index=False)
ds_b.to_csv("Hyperplane Dataset 30_2.csv", index=False)
ds_c.to_csv("Hyperplane Dataset 10_5.csv", index=False)
ds_d.to_csv("Hyperplane Dataset 30_5.csv", index=False)
```

```
def model(classifier, batch_size):
  print("Batch Size: {}".format(batch_size))
  start = time.time()
  stream_a = FileStream("Hyperplane Dataset 10_2.csv")
  stream_b = FileStream("Hyperplane Dataset 30_2.csv")
  stream_c = FileStream("Hyperplane Dataset 10_5.csv")
  stream_d = FileStream("Hyperplane Dataset 30_5.csv")
  streams = [stream_a, stream_b, stream_c, stream_d]
  accuracy = []
  inter_results = np.zeros([4, int(20000/batch_size)])
  for i in range(len(streams)):
    stream = streams[i]
    n_samples = 0
    corrects = 0
    dummy = 0
    while n_samples < 20000:
        X, y = stream.next_sample(batch_size)
        my_pred = classifier.predict(X)
        corrects += sum(y == my_pred)
        classifier = classifier.partial_fit(X, y)
        n_samples += batch_size
        inter_results[i][dummy] = corrects/n_samples
        dummy += 1
    accuracy.append(corrects/n_samples)
    print("Classifier's performance: {}".format(corrects/n_samples))
  runtime = time.time() - start
  print("Runtime: {}s".format(runtime))
  return accuracy, inter_results, runtime
# https://book.moa.cms.waikato.ac.nz/chapter_6.html/
# https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.la
```

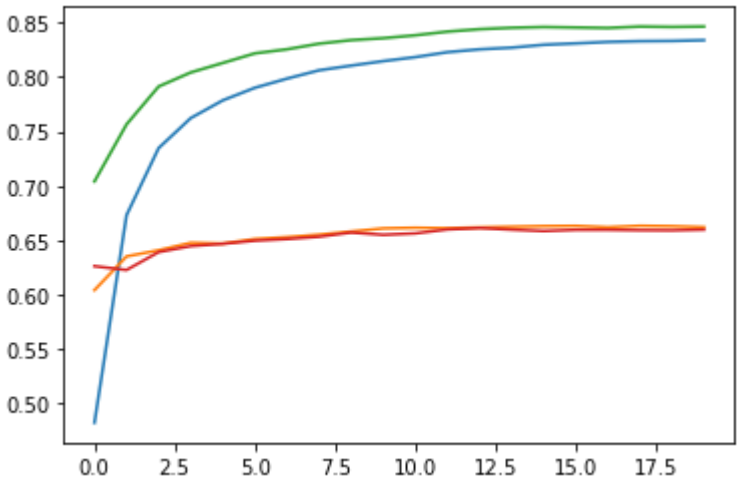
```
batch_arr = [1,100,1000]
knn_batch_acc = []
knn_batch_inter = []
knn_batch_time = []
print('KNN Classifier')

for batch_size in batch_arr:
  knn = mf.lazy.KNNClassifier(n_neighbors=141, max_window_size=2000)
  knn_acc, knn_inter, rt = model(knn, batch_size)
  knn_batch_acc.append(knn_acc)
  knn_batch_inter.append(knn_inter)
  knn_batch_time.append(rt)
```

```
KNN Classifier
Batch Size: 1
Classifier's performance: 0.846
```

```
Classifier's performance: 0.6634
Classifier's performance: 0.85155
Classifier's performance: 0.6592
Runtime: 198.61706948280334s
Batch Size: 100
Classifier's performance: 0.84435
Classifier's performance: 0.66405
Classifier's performance: 0.85285
Classifier's performance: 0.6612
Runtime: 23.743271589279175s
Batch Size: 1000
Classifier's performance: 0.83395
Classifier's performance: 0.6619
Classifier's performance: 0.8465
Classifier's performance: 0.65965
Runtime: 21.500442266464233s
```

```
1 plt.plot(knn_inter.T)
2
```

```
[<matplotlib.lines.Line2D at 0x7fb2a60bdfd0>,
 <matplotlib.lines.Line2D at 0x7fb2a60b9250>,
 <matplotlib.lines.Line2D at 0x7fb2a60b9410>,
 <matplotlib.lines.Line2D at 0x7fb2a60b95d0>]
```




```
1 batch_arr = [1,100,1000]
2 ht_batch_acc = []
3 ht_batch_inter = []
4 ht_batch_time = []
5 print('Hoeffding Tree Classifier')
6 for batch_size in batch_arr:
7   ht =  mf.trees.HoeffdingTreeClassifier()
8   ht_acc, ht_inter, rt = model(ht, batch_size)
9   ht_batch_acc.append(ht_acc)
10  ht_batch_inter.append(ht_inter)
11  ht_batch_time.append(rt)
```

```
Hoeffding Tree Classifier
Batch Size: 1
Classifier's performance: 0.84225
Classifier's performance: 0.6503
Classifier's performance: 0.81075
Classifier's performance: 0.6371
```

```
Runtime: 26.194802045822144s
Batch Size: 100
Classifier's performance: 0.84055
Classifier's performance: 0.65055
Classifier's performance: 0.8105
Classifier's performance: 0.6374
Runtime: 18.77136731147766s
Batch Size: 1000
Classifier's performance: 0.82425
Classifier's performance: 0.6514
Classifier's performance: 0.8083
Classifier's performance: 0.63585
Runtime: 17.305890560150146s
```

```
1
```

```
nb_batch_acc = []
nb_batch_inter = []
nb_batch_time = []
print('Naive Bayes Classifier')
for batch_size in batch_arr:
  nb =  mf.bayes.NaiveBayes()
  nb_acc, nb_inter, rt = model(nb, batch_size)
  nb_batch_acc.append(nb_acc)
  nb_batch_inter.append(nb_inter)
  nb_batch_time.append(rt)
```

```
Naive Bayes Classifier
Batch Size: 1
Classifier's performance: 0.88435
Classifier's performance: 0.65245
Classifier's performance: 0.8089
Classifier's performance: 0.64395
Runtime: 12.538081407546997s
Batch Size: 100
Classifier's performance: 0.883
Classifier's performance: 0.6521
Classifier's performance: 0.80885
Classifier's performance: 0.64395
Runtime: 8.065203666687012s
Batch Size: 1000
Classifier's performance: 0.8667
Classifier's performance: 0.6518
Classifier's performance: 0.808
Classifier's performance: 0.64355
Runtime: 7.887600898742676s
```

```
1
```

Ensemble

```
batch_size = 1
stream_a = FileStream("Hyperplane Dataset 10_2.csv")
stream_b = FileStream("Hyperplane Dataset 30_2.csv")
stream_c = FileStream("Hyperplane Dataset 10_5.csv")
```

```
stream_c = FileStream("Hyperplane Dataset 10_5.csv")
stream_d = FileStream("Hyperplane Dataset 30_5.csv")
streams = [stream_a, stream_b, stream_c, stream_d]
start = time.time()
accuracy = []
inter_results = np.zeros([4, int(20000/batch_size)])
for i in range(len(streams)):
  stream = streams[i]
  knn_mv = mf.lazy.KNNClassifier(n_neighbors=141, max_window_size = 2000)
  nb_mv = mf.bayes.NaiveBayes()
  ht_mv = mf.trees.HoeffdingTreeClassifier()
  n_samples = 0
  corrects = 0
  dummy = 0
  while n_samples < 20000:
      X, y = stream.next_sample()
      knn_pred = knn_mv.predict(X)
      ht_pred = ht_mv.predict(X)
      nb_pred = nb_mv.predict(X)
      if (knn_pred[0]+ht_pred[0]+nb_pred[0]) >= 2:
        my_pred = 1
      else:
        my_pred = 0
      corrects += sum(y == my_pred)
      knn_mv = knn_mv.partial_fit(X, y)
      nb_mv = nb_mv.partial_fit(X, y)
      ht_mv = ht_mv.partial_fit(X, y)

      n_samples += batch_size
      inter_results[i][dummy] = corrects/n_samples
      dummy += 1
  accuracy.append(corrects/n_samples)
  print("Ensamble Classifier's performance: {}".format(corrects/n_samples))
runtime = time.time() - start
print("Runtime: {}s".format(runtime))



# https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.la
```

```
Ensamble Classifier's performance: 0.8744
Ensamble Classifier's performance: 0.6873
Ensamble Classifier's performance: 0.87385
Ensamble Classifier's performance: 0.6838
Runtime: 230.64989948272705s
```

```
1
```