PEARL

Probabilistic Exact Adaptive Random Forest with Lossy Counting for Data Streams

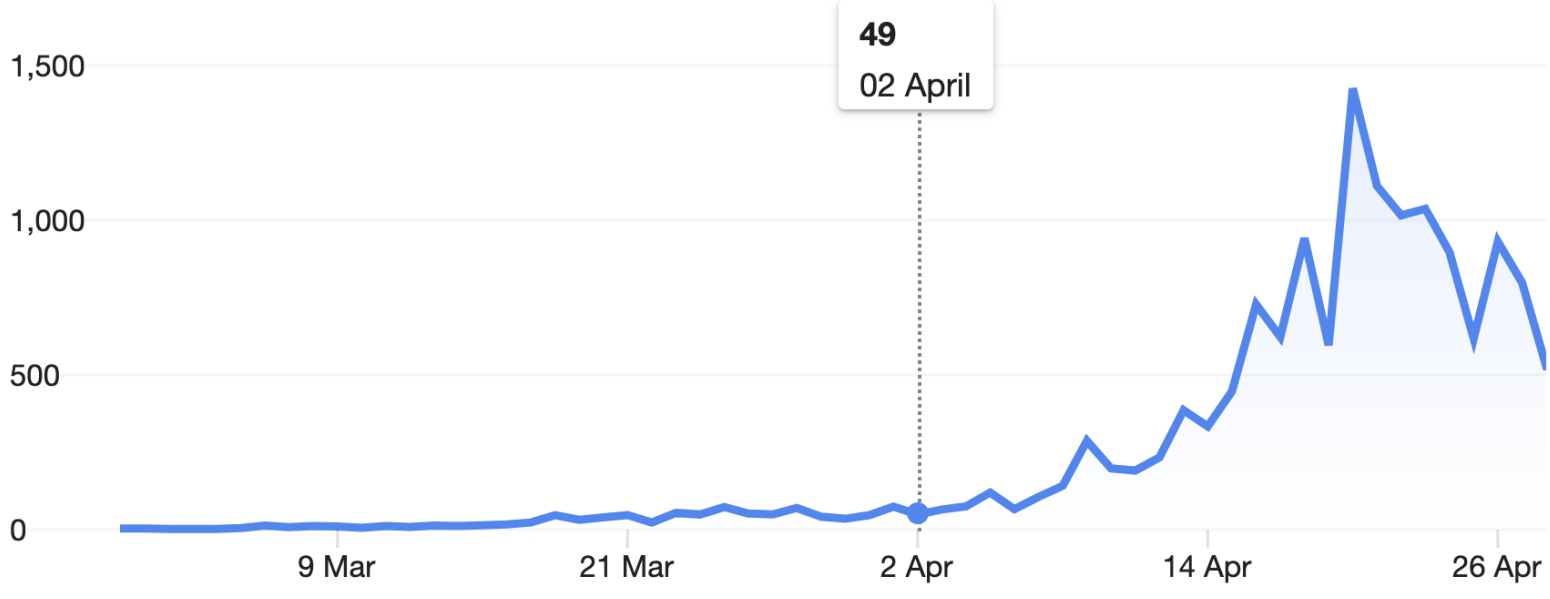
Ocean Wu, Yun Sing Koh, Gillian Dobbie, Thomas Lacombe

The University of Auckland

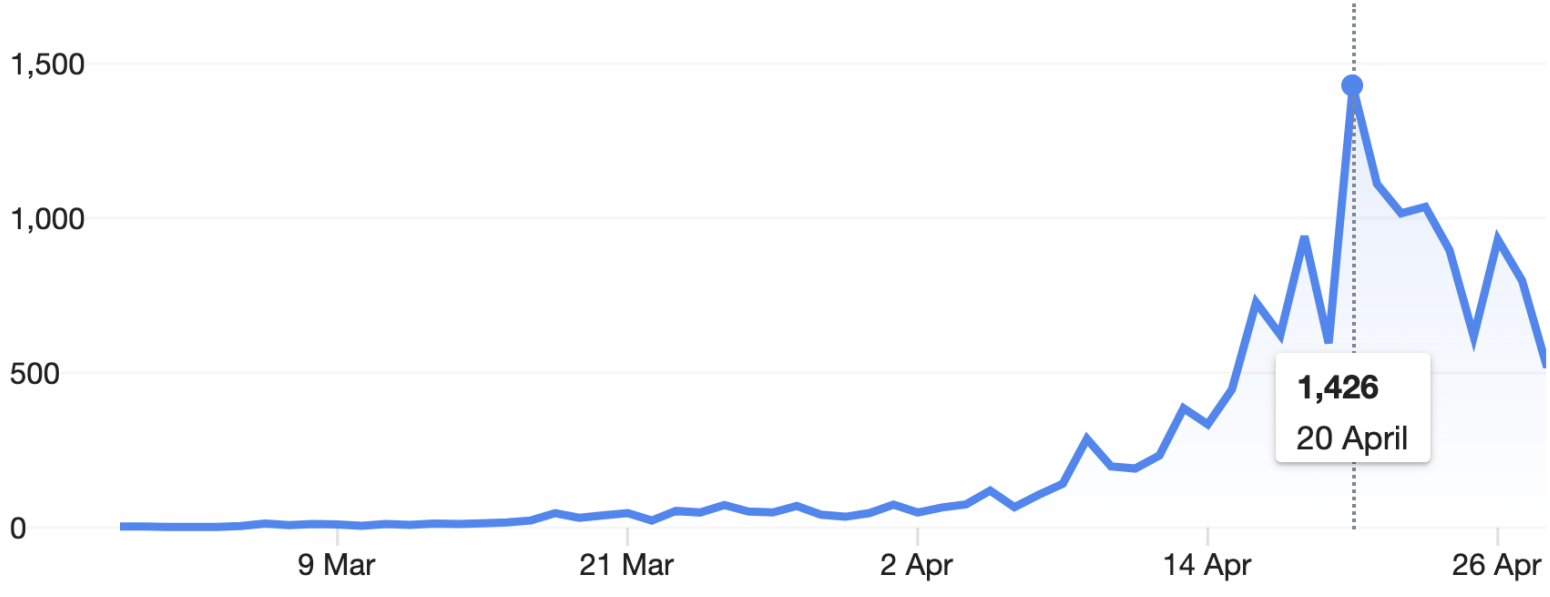
A continuous stream of data...

- Infinite
- Ordered
- Generated in real time

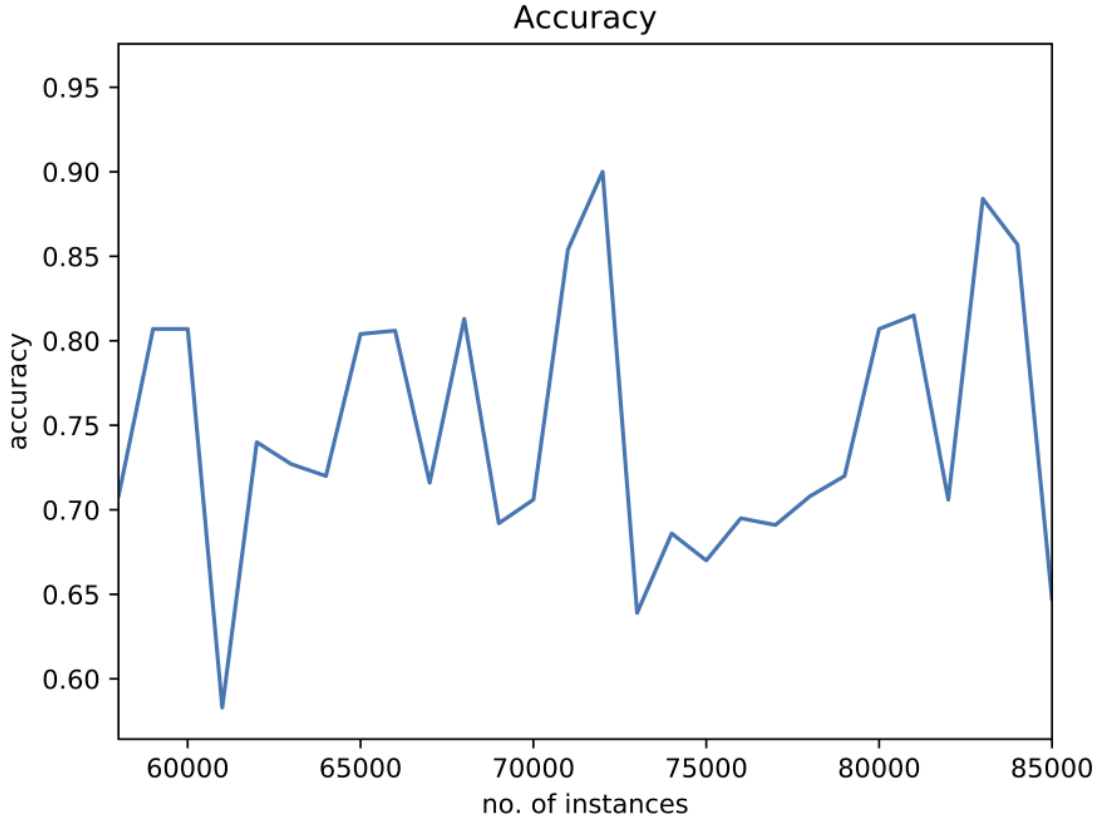
New cases of coronavirus in Singapore



New cases of coronavirus in Singapore



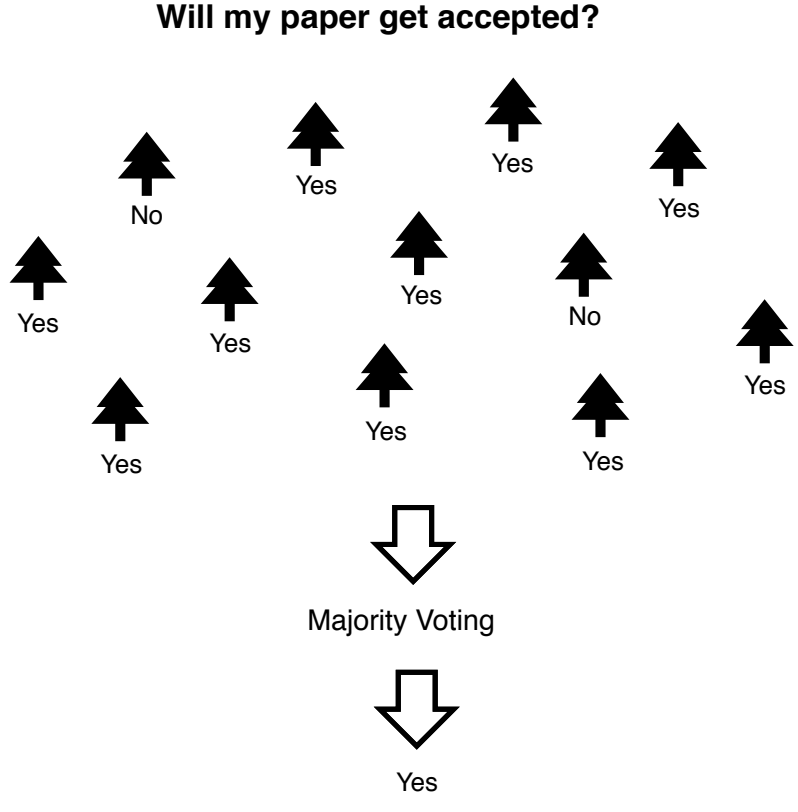
Concept Drift



Other Examples of Concept Drift

- Increase sales of ice-cream in summer.
- Self-driving cars detect pedestrian crossing illegally.
- Adversary actions such as bank frauds.
- ...

Random Forest



State-of-the-Art: Adaptive Random Forest

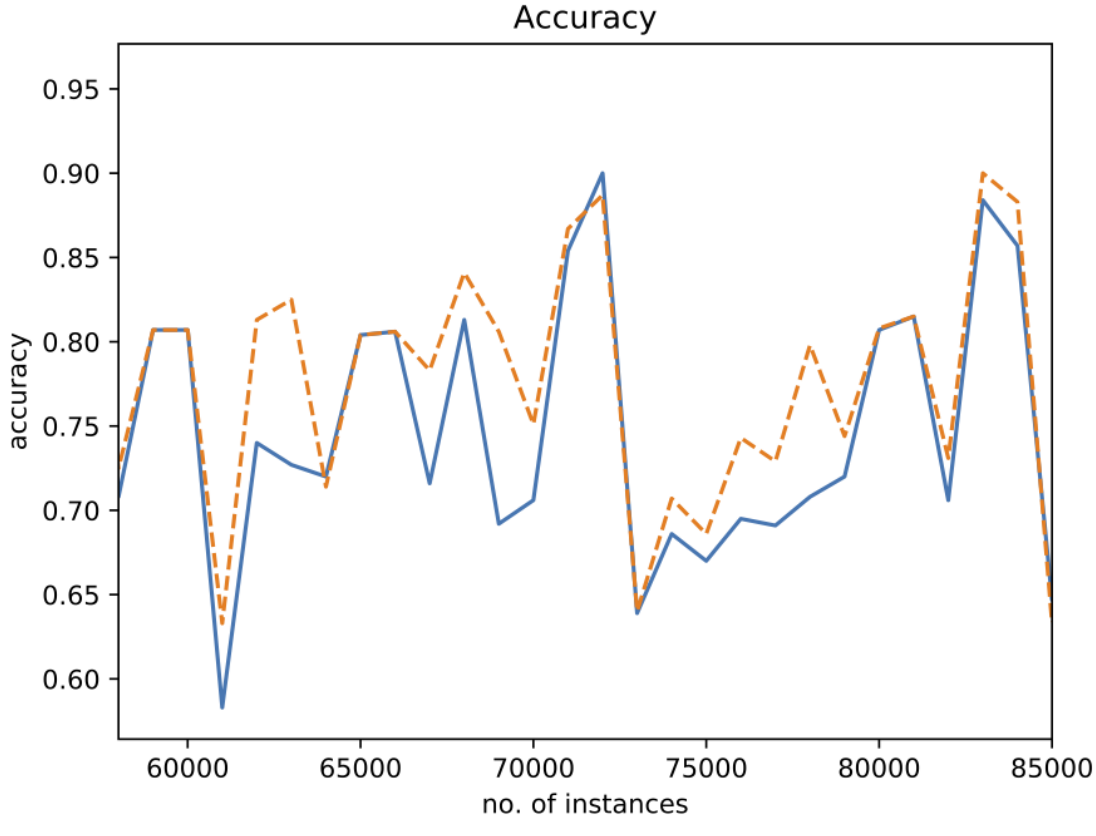
Foreground Trees

- Get trained and participate in voting
- Each tree has two *concept drift detectors* for detecting
    - Drift warning
    - Actual drift

Background Trees

- Drift warning detected - starts training
- Actual drift detected - replaces the foreground tree

State-of-the-Art: Adaptive Random Forest



PEARL

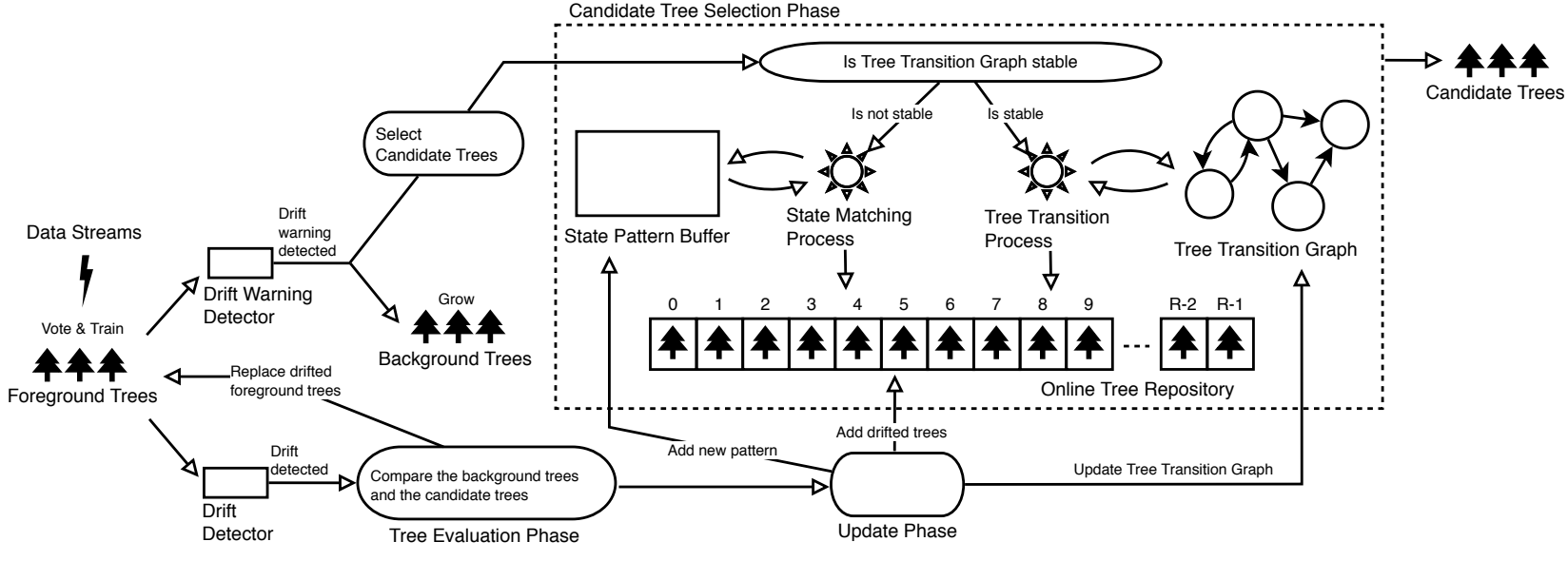
Foreground trees does not get discarded when they are replaced by their background tree.

Instead they are stored in an *online tree repository*.

Candidate Trees

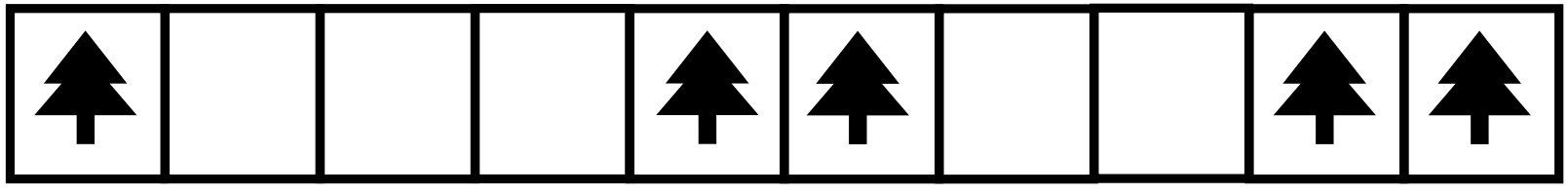
- A subset of the trees from the online tree repository
- The subset gets updated when drift warnings are detected on the foreground trees

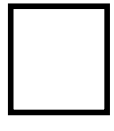
PEARL Architecture



State Matching Process

Pattern Construction



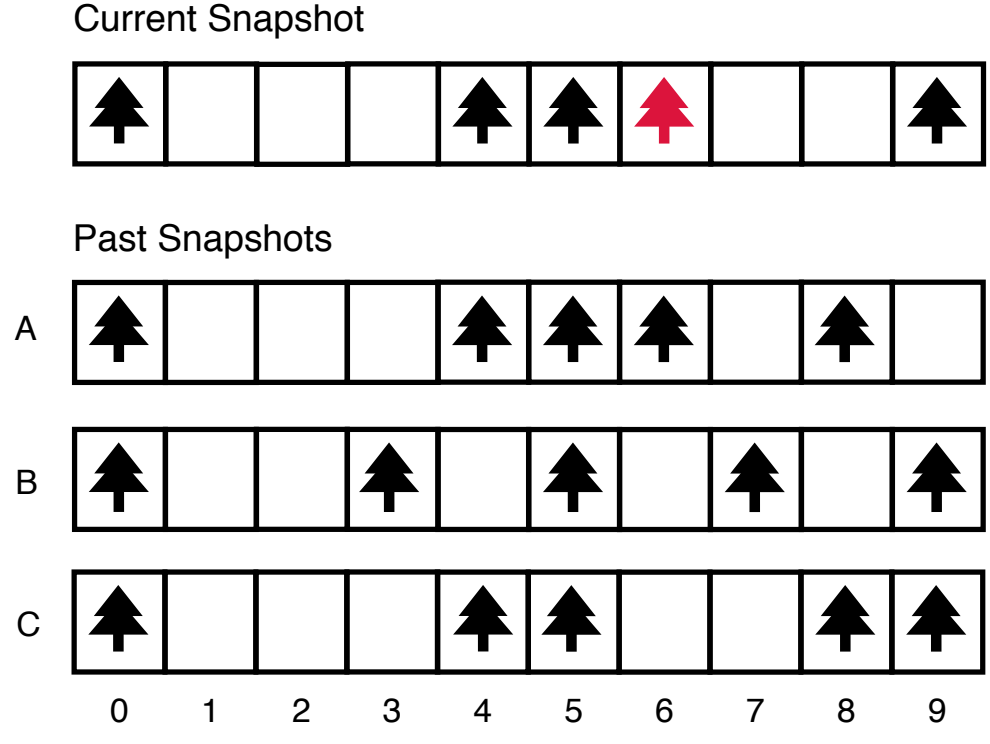


A slot in the Online Tree Repo

The tree in the slot is used as a Foreground Tree

State Matching Process

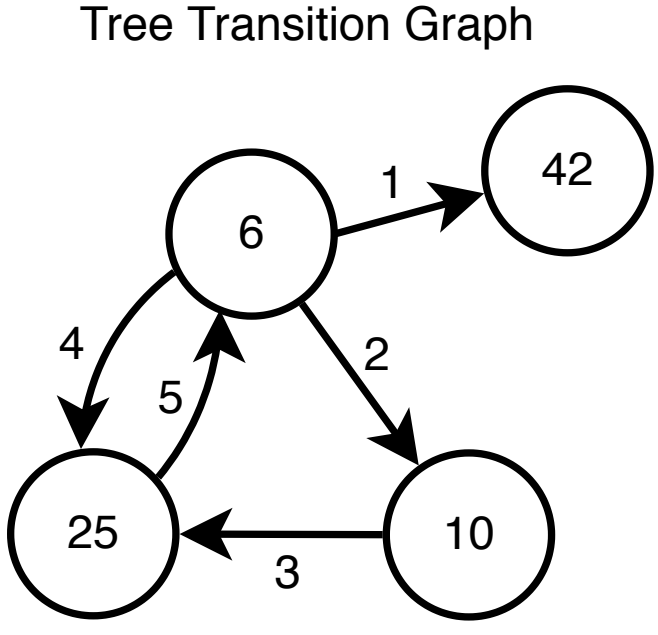
Exact Pattern Matching

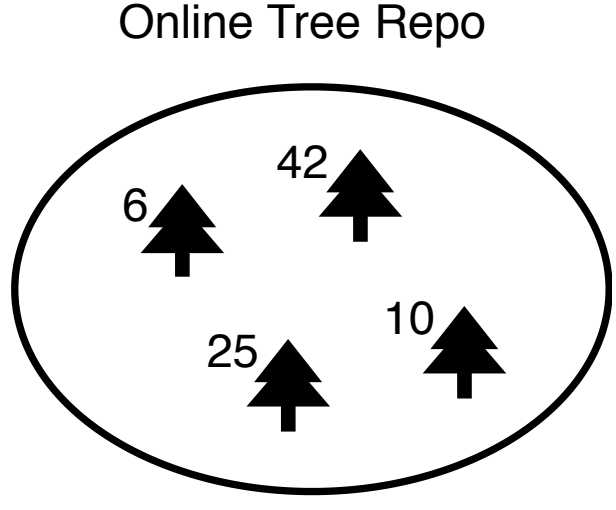


- Works well until there are too many patterns

Tree Transition Process

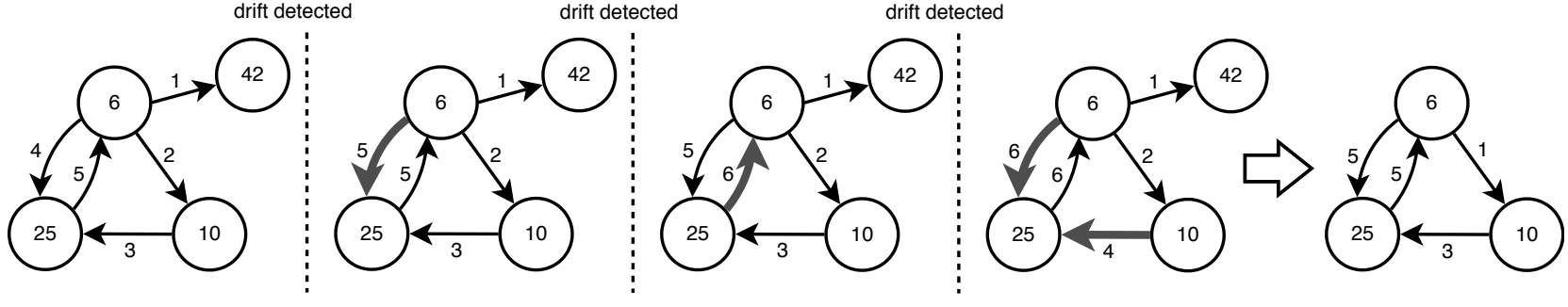
Probabilistic Graphical Model



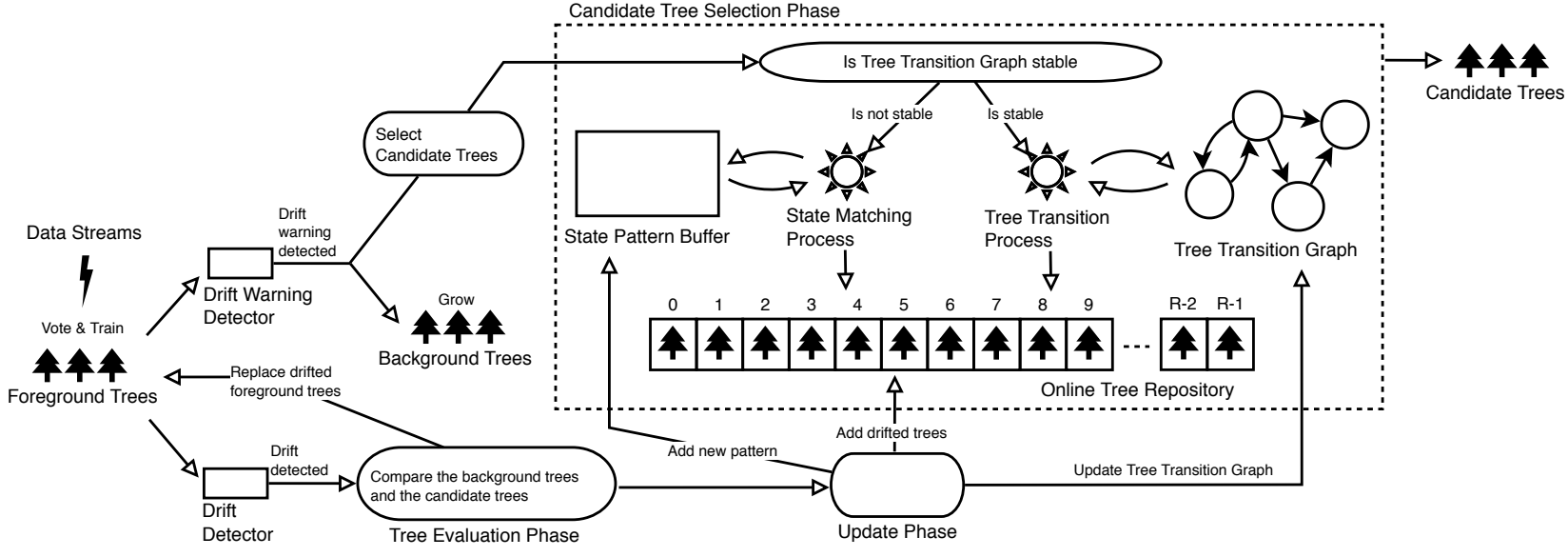


- Wrong transitions add noise
- Some trees in the online tree repo may loose relevancy as data stream evolves

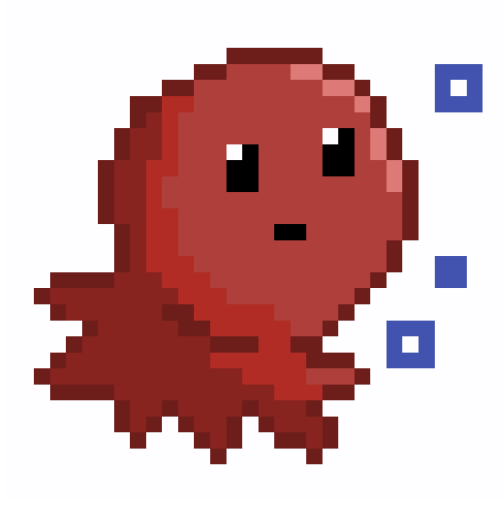
Lossy Counting



Contributions



- Introduced candidate trees to the Adaptive Random Forest
- Developed a novel framework for selecting and applying the candidate trees
    - Exact technique: pattern matching
    - Probabilistic technique: graphical model with Lossy Counting



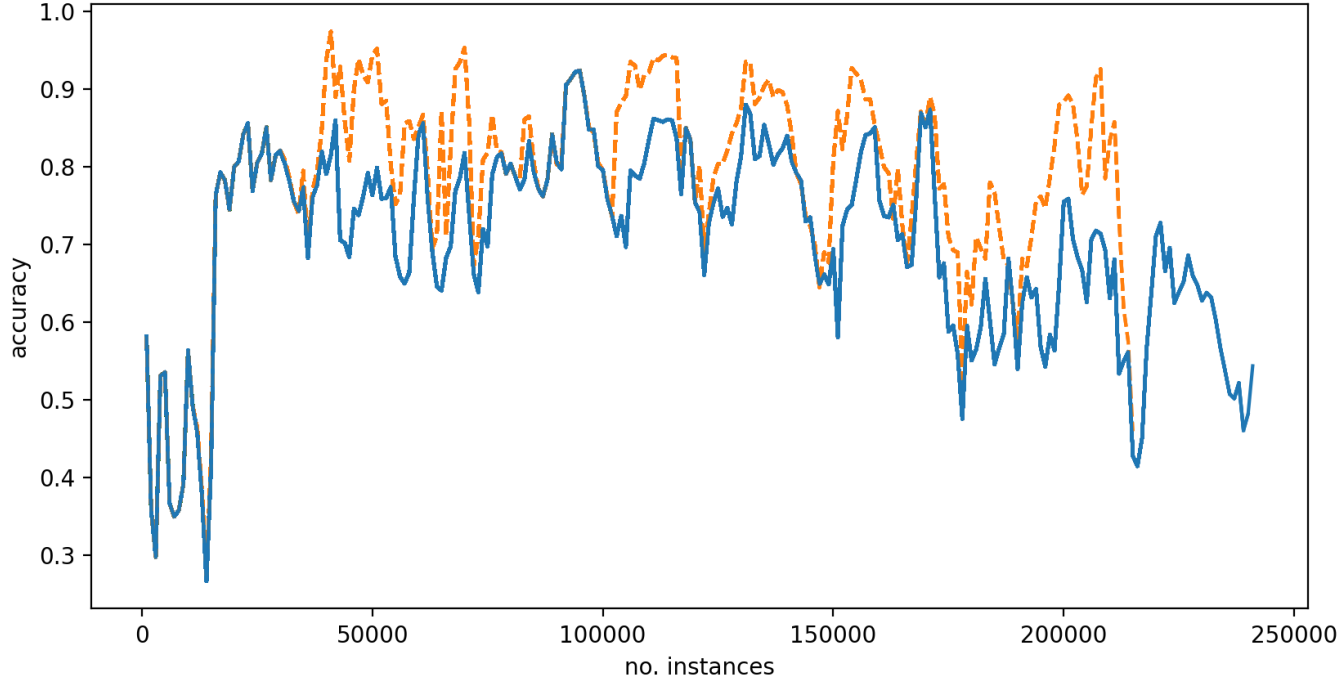
In [3]:

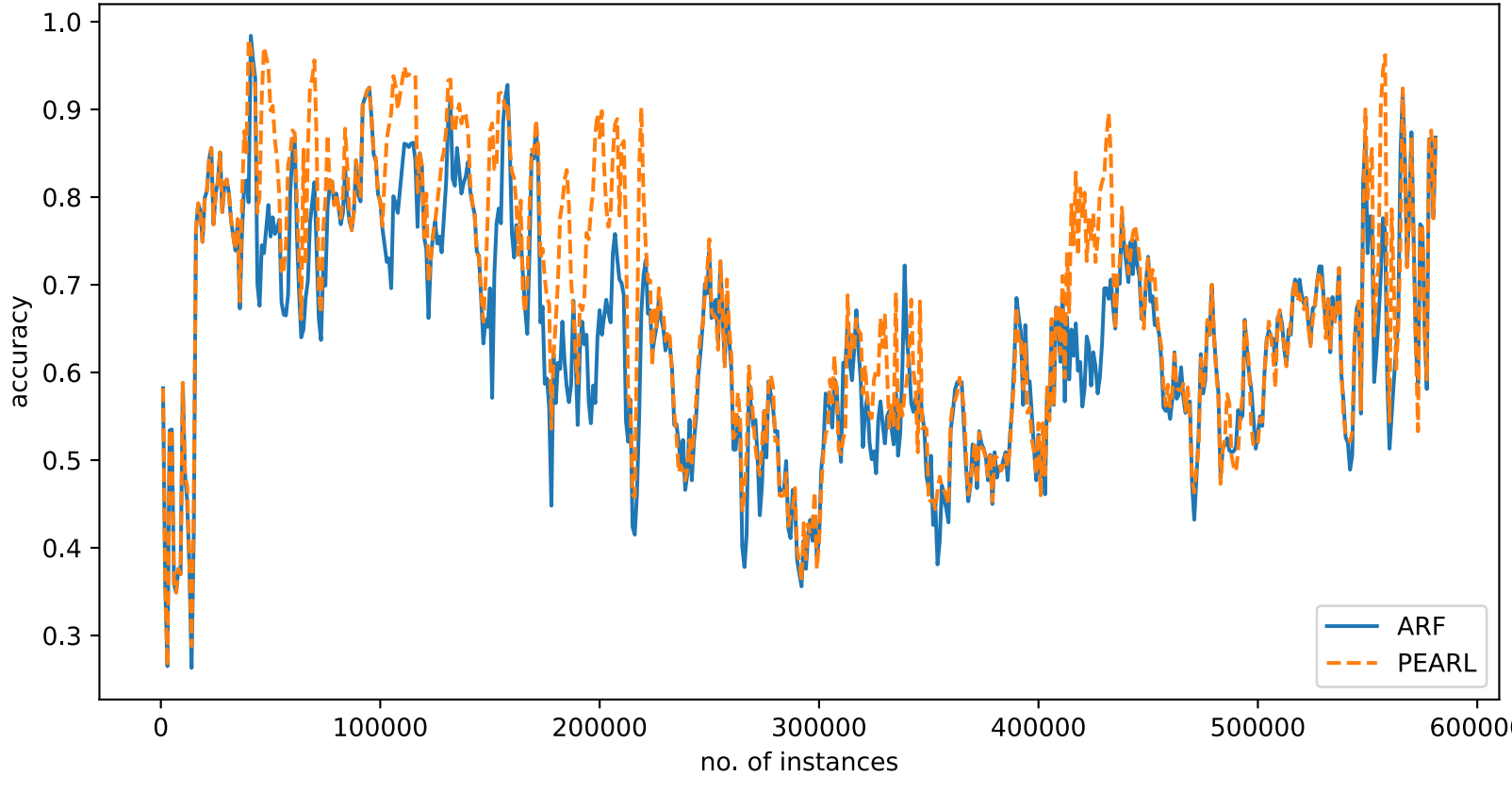
```
%matplotlib notebook

from matplotlib import pyplot as plt
from matplotlib import animation

from skika.ensemble import adaptive_random_forest, pearl


class evaluate(object):

    def __init__(self, classifier):
        self.accuracy = 0
        self.num_instances = 0
        self.classifier = classifier

        self.classifier.init_data_source("data/covtype.arff");

    def __call__(self):
        correct = 0
        sample_freq = 1000

        for count in range(0, sample_freq):
            if not self.classifier.get_next_instance():
                break

            # test
            prediction = self.classifier.predict()

            actual_label = self.classifier.get_cur_instance_label()
            if prediction == actual_label:
                correct += 1

            # train
            self.classifier.train()

            self.classifier.delete_cur_instance()
```

```
        self.accuracy = correct / sample_freq
        self.num_instances += 1000

        return self.num_instances, self.accuracy


num_trees = 60
max_num_candidate_trees = 120
repo_size = 9000
edit_distance_threshold = 90
kappa_window = 50
lossy_window_size = 100000000
reuse_window_size = 0
max_features = -1
bg_kappa_threshold = 0
cd_kappa_threshold = 0.4
reuse_rate_upper_bound = 0.18
warning_delta = 0.0001
drift_delta = 0.00001
enable_state_adaption = True
enable_state_graph = True

arf_classifier = adaptive_random_forest(num_trees,
                                        max_features,
                                        warning_delta,
                                        drift_delta)
arf = evaluate(arf_classifier)

pearl_classifier = pearl(num_trees,
                         max_num_candidate_trees,
                         repo_size,
                         edit_distance_threshold,
                         kappa_window,
                         lossy_window_size,
                         reuse_window_size,
                         max_features,
                         bg_kappa_threshold,
                         cd_kappa_threshold,
```

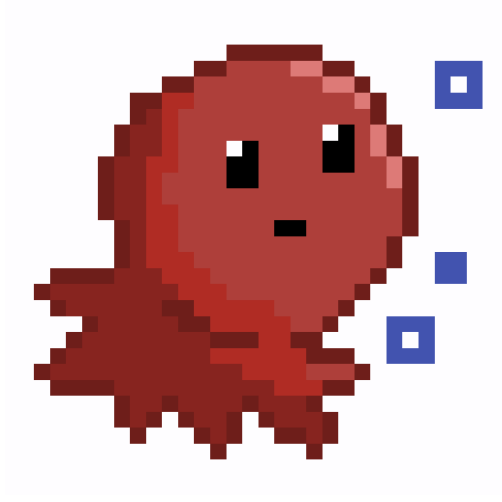
```
                            reuse_rate_upper_bound,
                            warning_delta,
                            drift_delta,
                            enable_state_adaption,
                            enable_state_graph)
pearl = evaluate(pearl_classifier)

fig = plt.figure()

x_arf = []
y_arf = []
x_pearl = []
y_pearl = []

max_samples = 580000

def frames_arf():
    for i in range(max_samples):
        yield arf()

def animate_arf(args):
    x_arf.append(args[0])
    y_arf.append(args[1])
    return plt.plot(x_arf, y_arf, color='C0', linestyle='-', label='ARF')

def frames_pearl():
    for i in range(max_samples):
        yield pearl()

def animate_pearl(args):
    x_pearl.append(args[0])
    y_pearl.append(args[1])
    return plt.plot(x_pearl, y_pearl, color='C1', linestyle='--', label='PEARL')


anim_arf = animation.FuncAnimation(fig, animate_arf, frames=frames_arf, interval=1
000)
anim_pearl = animation.FuncAnimation(fig, animate_pearl, frames=frames_pearl, inte
```

```
rval=1000)

plt.xlabel('no. instances')
plt.ylabel('accuracy')

F = plt.gcf()
Size = F.get_size_inches()
F.set_size_inches(Size[0]*1.5, Size[1]*1, forward=True)
plt.show()
```




1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
accuracy
0
100000
200000
300000
400000
500000
600000
no. of instances
ARF
PEARL

scikit-ika



An open source real-time adaptive predictive system for evolving data streams. [Homepage (https://scikit-ika.github.io)](https://scikit-ika.github.io)

Questions