Project 2 Week 1

Figure 1

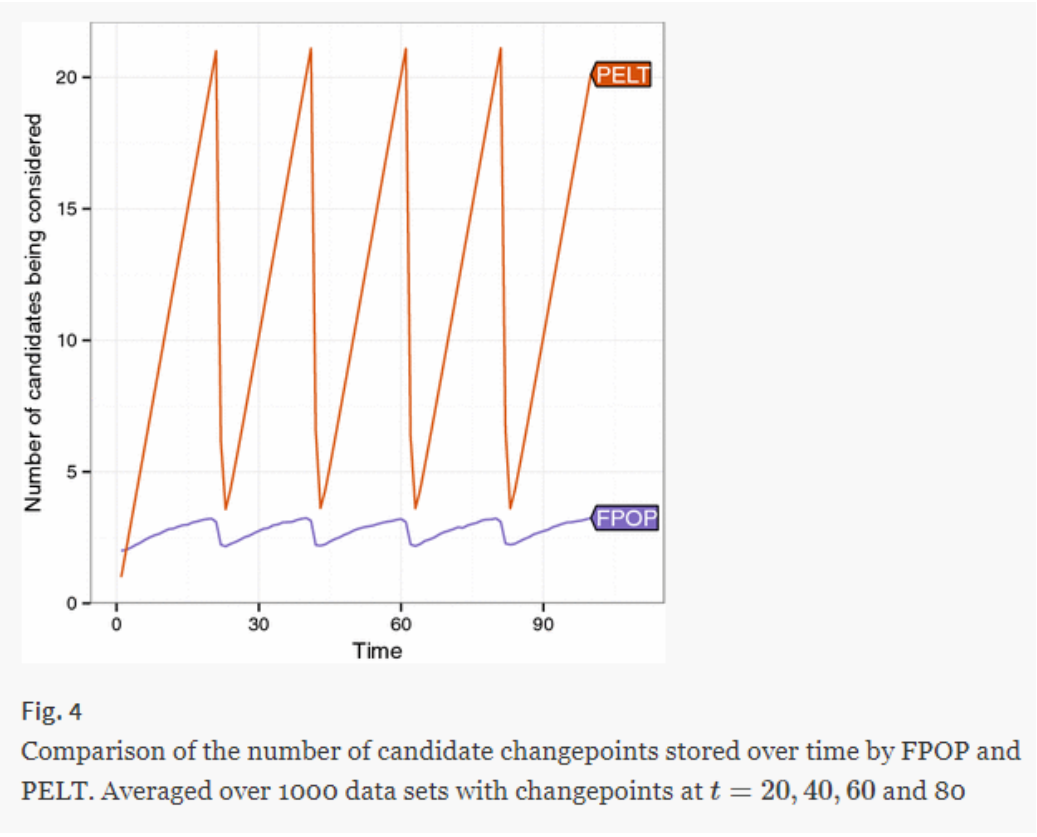

Fig. 4
Comparison of the number of candidate changepoints stored over time by FPOP and PELT. Averaged over 1000 data sets with changepoints at $t = 20, 40, 60$ and 80

Citation

Toby Hocking, Guillem Rigaill, Paul Fearnhead et. al. On optimal multiple changepoint algorithms for large data Fig. 4

Problem Setting

For this figure we have to implement PELT (Pruned exact linear time) algorithm for change point detection and keep track the number of change points being considered at each time step for simulated data with specified change points. We then perform the same procedure for fpop and then plot the results to see the comparison.

The input to the algorithm would be simulated data sets of size 100 with changepoints and 20,40,60 and 80. The output would be a plot which shows the comparison with fpop.

Data Sources

For this figure we will use simulated data of size 100 which can be generated using **rnorm** function in R that generates a dataset with specified size from a normal distribution with given mean.

Algorithm

We need to implement the PELT algorithm to get the number of changepoints being considered at each time step. The PELT algorithm improved the classical optimal partitioning algorithm by limiting the set of potential previous changepoints. If function **F** denotes the optimal cost of partitioning and **C** be the cost function then for data point t if **F(s) + C(Y(s+1:t)) + K > F(t)** where, **Y(s+1:t)** is the segment starting at data point s+1 and ending at t holds for any value of K then at any future time **T > t, s** can never be the optimal location of change point and thus can be pruned.

The pseudo code would look as follows:

1. **Input**: set of data points(y1, y2, …, yn), penalty constant B, constant K as described above
2. Set F(0) = -B, R1 = {0}
3. for T in 1, …, n
4. Calculate F(T') = min [F(T) + C(Y(T+1:T') + B] for all T in RT'
5. Let T1 be the minimum value obtained from step 4 corresponding to position of last change point.
6. Set R(T'+1) = {T for all T in RT' U {T'} : F(T) + C(Y(T+1:T') + K < F(T')}
7. **Output:** The sets Ri will give the candidates considered at each time step i

To compare this with fpop we can utilize the implementation from fpop package and read the documentation to update the code to store the candidates considered at each time step.

Implementing these algorithms will help me get better understanding of pruning methods for time-series data which might help me in my project for labelled change point detection.

Figure 2

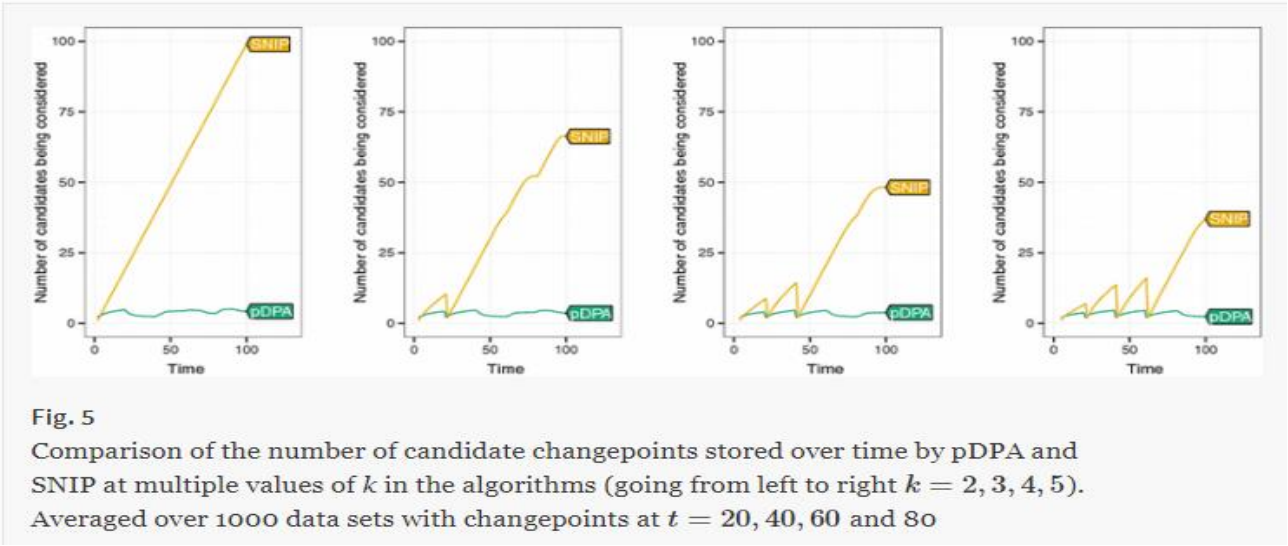

Fig. 5
Comparison of the number of candidate changepoints stored over time by pDPA and SNIP at multiple values of k in the algorithms (going from left to right $k = 2, 3, 4, 5$). Averaged over 1000 data sets with changepoints at $t = 20, 40, 60$ and 80

Citation

Toby Hocking, Guillem Rigaill, Paul Fearnhead et. al. On optimal multiple changepoint algorithms for large data Fig. 5

Problem Setting

For this figure we have to implement SNIP (Segment Neighborhood Inequality Pruning) algorithm for constrained change point detection with specified maximum number of change points **K** and keep track the number of change points being considered at each time step for simulated data with specified change points. We then perform the same procedure for pDPA and then plot the results to see the comparison.

The input to the algorithm would be simulated data sets of size 100 with changepoints and 20,40,60 and 80 and k = 2,3,4 and 5 where k is the constant that satisfies the pruning condition. The output would be a plot which shows the comparison with pDPA.

Data Sources

For this figure we will use simulated data of size 100 which can be generated using **rnorm** function in R that generates a dataset with specified size from a normal distribution with given mean.

Algorithm

We need to implement the SNIP algorithm to get the number of changepoints being considered at each time step. The SNIP algorithm works exactly as the PELT with the only difference that the max number of change points are specified.

The pseudo code would look as follows:

1. **Input**: set of data points(y1, y2, …, yn), penalty constant B, constant K = max no. of changepoints, k is the constant that satisfies the inequality condition for pruning
2. Set F(0) = -B, R1 = {0}
3. for T in 1, …, K
4. Set R (T,T+1) = {T}
5. For t in T+1, …, n
6. Calculate C (T,t) = min [C(T-1, v) + C(Y(v+1:t))] for all v in R (T, t)
7. Set R (T,t+1) = { v in {R (T,t) U {t}} : C (T-1,v) + C(Y(v+1:t)) + k < C (T-1,t)}
8. Set T' (T,1) = min [C (T-1,v) + C(Y(v+1:n))]
9. **Output:** Sets R (k, i) will give the candidates considered at each time step I with k change points

Where C (k, t) denotes the optimal cost of partitioning data till point t with k change points and R (k, t) denotes the candidate set for point t with k change points.

To compare this with pDPA we can utilize the implementation from Segmentor3IsBack package which provides implementations of various constraint change point detection problems and read the documentation to update the code to store the candidates considered at each time step.

Implementing SNIP will give me better understanding of inequality based pruning for constrained change point detection.