Week 10 Writeup

# RECAP: Week 9 Summary

**Main improvements:** Finished clustering-classification-clustering pipeline & evaluated/tweaked performance. Also created direct regression to predict KS statistic between unseen & seen edges

1. **[MON]** – Complete pipeline: convert predicted class 0 (dissimilar edge) & class 1 (similar edge) into clusters using predicted probability of class 0.
   1. Very similar clusters to edge length clustering on AAF map
   2. Similar clusters to KS clustering on TSC map
2. **[TUE & WED]** – Compare classification method to baseline methods (edge length clustering & KS clustering)
   1. Only very small improvement on naïve method of edge length clustering (by ~0.005 mean KS)
   2. Worse than KS clustering (by ~0.05 mean KS) – this is expected since KS clustering requires data
   3. Method: calculate KS of each edge in a cluster vs all other edges in a cluster & take mean.
      1. Either use 2 sample KS using raw data (higher KS but same differences in performance between methods)
      2. Or use KS test involving fitted edges (lower mean KS)
3. **[THU]** – Tweak classification models to try and combat overfitting 🡺 small performance improvements (~1-2%)
   1. Classification models perform at 0.70-0.75 accuracy, 0.75-0.85 AUC on unseen test data
   2. Worse performance is expected compared to the training set (~0.9 accuracy, 0.95 AUC)
   3. Maybe the performance is not better than edge length clustering because there are 3 steps, each with misclassification/misclustering rates?
4. **[FRI]** – Try different clustering approaches (Affinity Propagation, DBSCAN, OPTICS, Spectral Clustering, HAC using max distance instead of n\_clusters)
   1. No significant improvements: still problems with “anomalous” edges. Out of the methods tried, DBSCAN & Spectral Clustering are the best.
   2. Affinity Propagation & OPTICS are unable to find clusters
   3. DBSCAN has large variance in clusters for small changes in hyperparameters
   4. Spectral clustering also asks for a defined number of clusters
   5. **TO DO:** try DIANA
5. **[SUN]** – Regression approach
   1. Predict a KS score from spatial features
   2. Tried: OLS (Lasso, Ridge), SVR, Random Forest, AdaBoost, Gradient Boost
   3. Evaluate by taking mean KS of n similar edges in training map with each edge in test map

TO DO:

1. Baseline for Regression
2. Use regression KS scores for clustering on test map. Or between train & test maps
3. Mergefit method for classification pipeline & regression
4. Use DIANA for clustering
5. Is more data required?

Next steps:

1. FUTURE: Create maps with interesting properties
2. Evaluation: create new map, no data collection phase, start with an informative prior & refine
   1. How much better is it than MLE with data available (time = distance / speed)
   2. Or Bayesian method (compute mean & variance of prior)
   3. Transfer data from previous experience: we want this to perform better at the beginning
      1. Just pick according to edge length
      2. Use binary classifier
      3. Use KS regression
      4. Pick random edge/ same edge for all
   4. In the long term, these methods should all be the same
   5. Fit distributions vs the empirical data & see how the KS changes
3. FUTURE: Treat slowing/speeding up as contexts: origin/final points
4. FUTURE: Try and regress the properties of the prior
5. Documentation + code

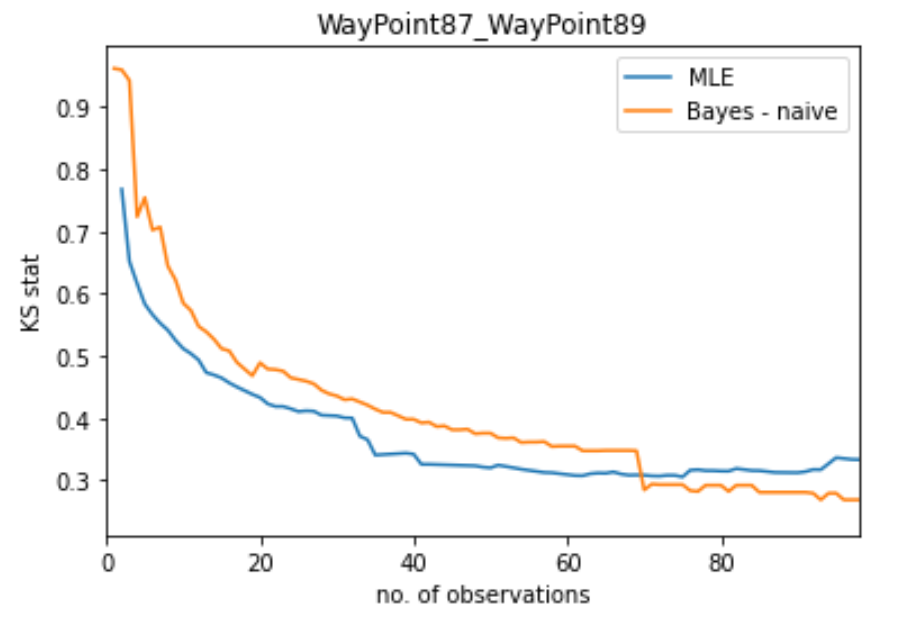
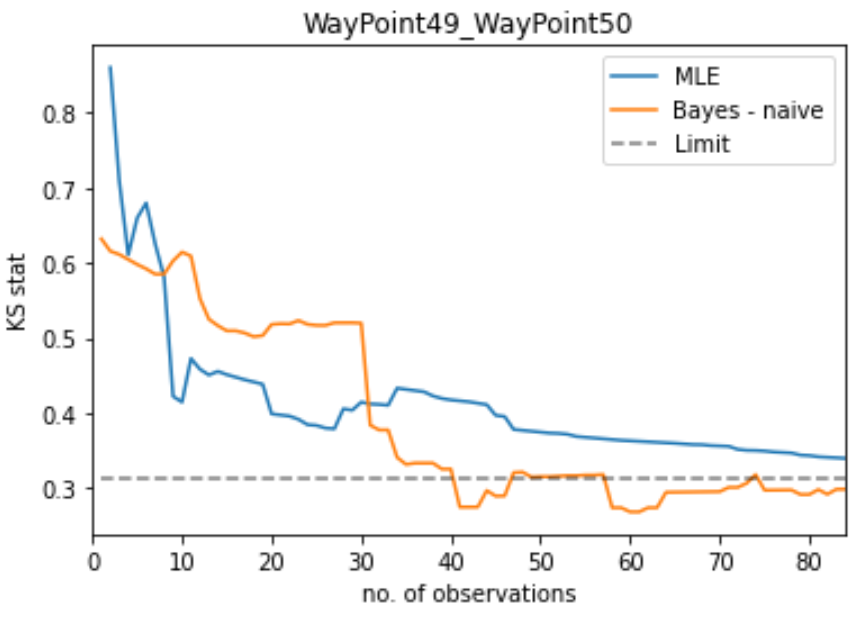
# Plan for Week 10

1. **[TUE] – Evaluation Part 1:** create framework to compare MLE & Bayesian method (naïve prior) as datapoints are fed into the system. Create some helper docs/readmes that make github/writeups more user-friendly
2. **[WED] – Evaluation Part 2:** Add random picking & picking according to closest edge length
3. **[THU] – Evaluation Part 3:** Add binary classifier to pick according to edges with the highest probability of similarity (i.e. probability of class 0)
4. **[FRI] – Evaluation Part 4:** Add KS regressor to pick according to edges with lowest predicted KS

# TUE: MLE & Bayesian (naïve)

Both methods have fairly quick convergence to the minimum KS limit. However, this depends on the edge and also varies between runs (since I am using data in a random order)

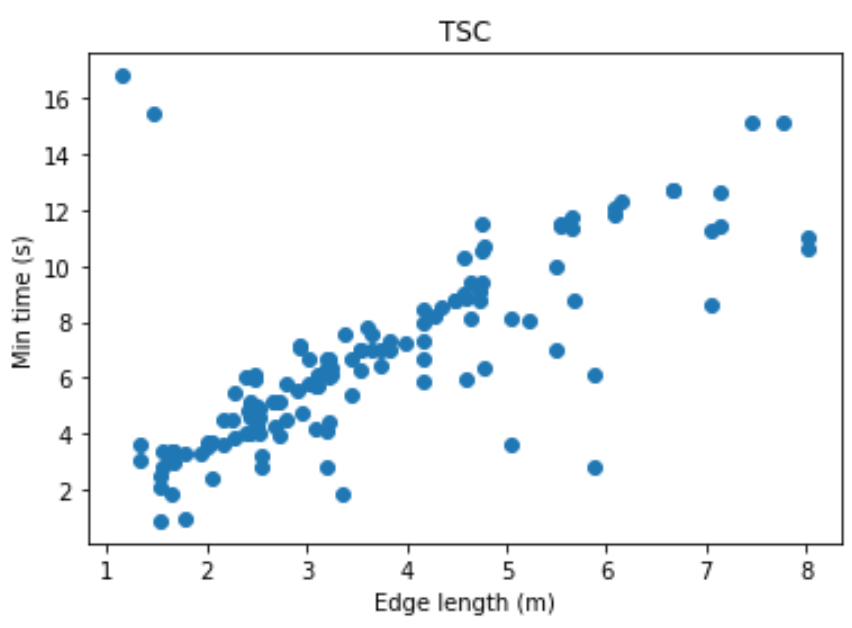
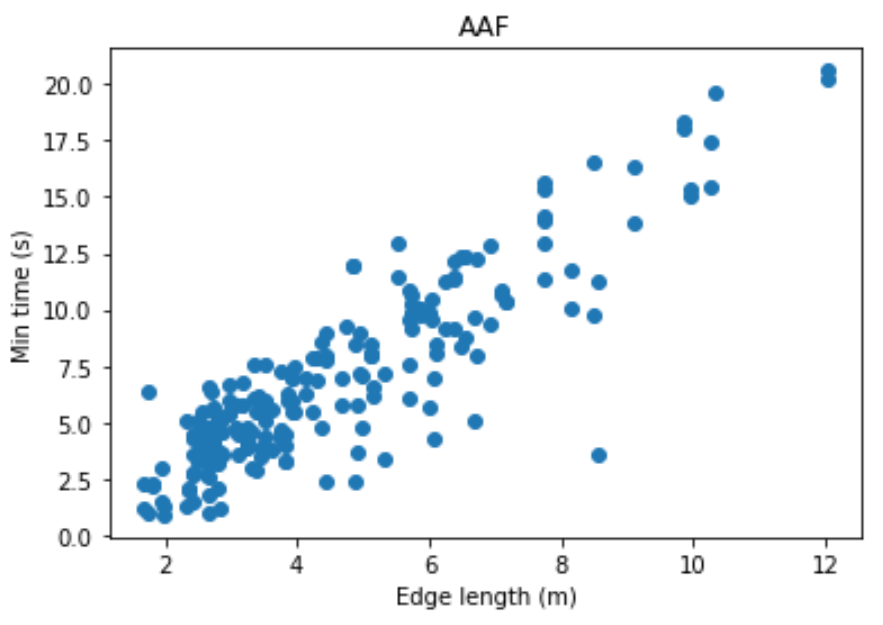
* For WP87\_WP89 (attachment 1): this is about 50-70 observations.
* For WP49\_WP50 (attachment 2): this can be as fast as 10-20 observations.



For the remaining methods, **the goal** is to start with a lower KS value at 0 observations and require fewer additional observations to converge to the limit.

A quick note about my **method for finding the offset parameter** for the lognormal:

* Lognormal distribution has 3 params: offset, shape (std deviation of LN(t) ), scale ( mean of t )
* To find the offset term, I originally took the minimum observed duration in the entire dataset. However, this is not reflective of the sequential way of collecting data.
* For the current implementation, I use the minimum observed duration that has been "seen" by the system.
* We then update the offset (which also means that we need to retrain the shape & scale) when we see a shorter duration than our previous minimum (with a small buffer).
* This seems to work well in practice since we only need 5-8 such retrainings for ~1000 datapoints (for the naive bayesian method this takes ~5 seconds in total)
* An alternative approach to this is to guess a minimum time based on the edge length and average speed.
  + This could be viable since the edge length & minimum observed time are positively correlated for AAF & TSC maps (see below).
  + However, this again assumes knowledge of durations since AAF has max speed of (0.75 ± 0.35 m/s) and TSC has max speed of (0.6 ± 0.25 m/s) although the maps use the same robot

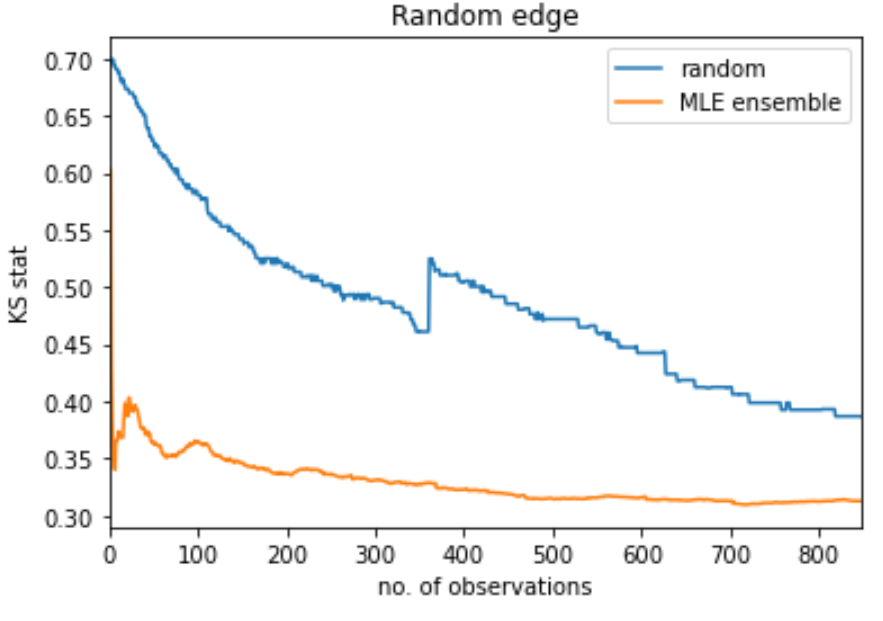
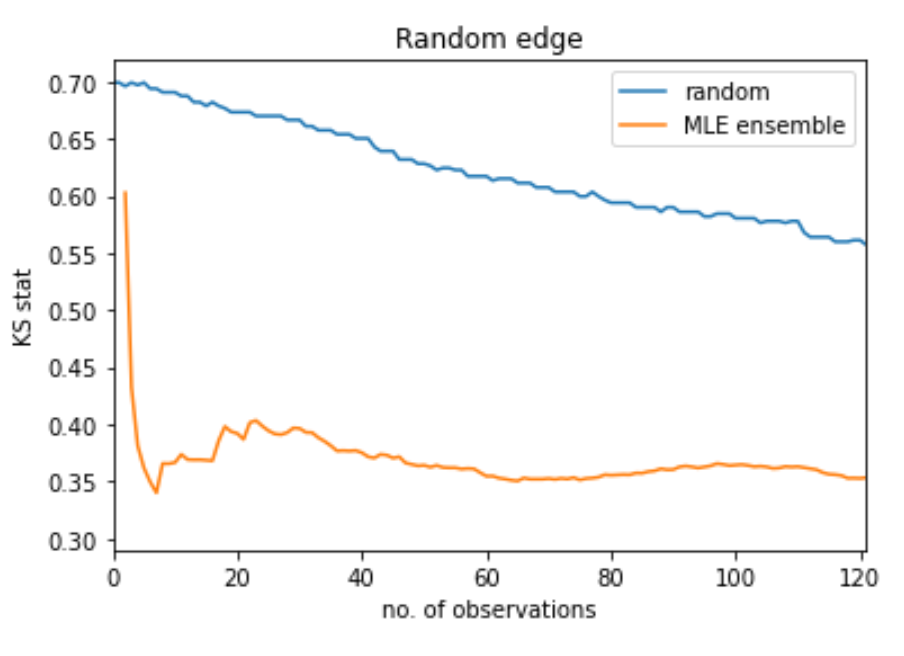


# WED: Random Edge & Closest Edge length

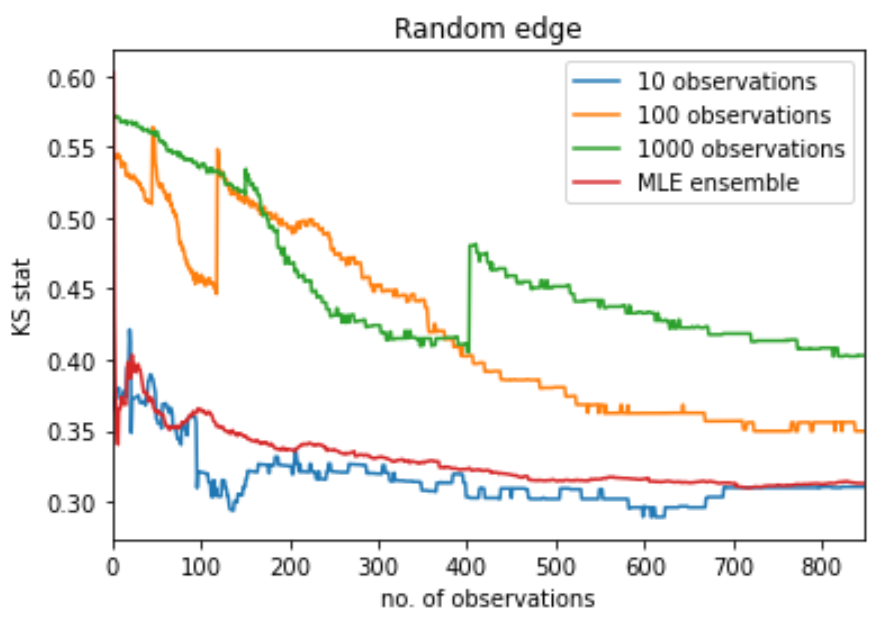
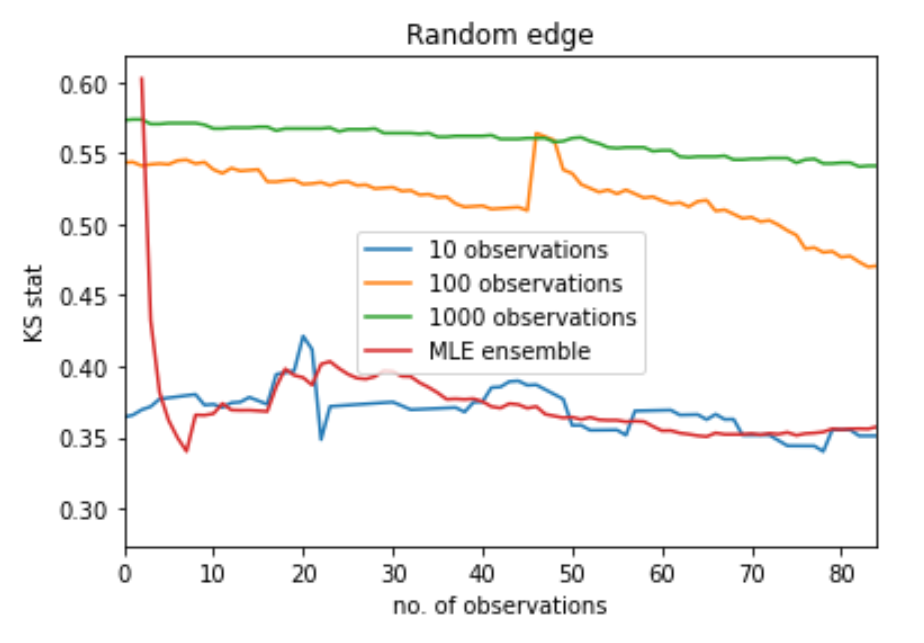
Note: the below graphs are for a test edge of “WP49\_WP50” in the AAF map

## Random Edge:

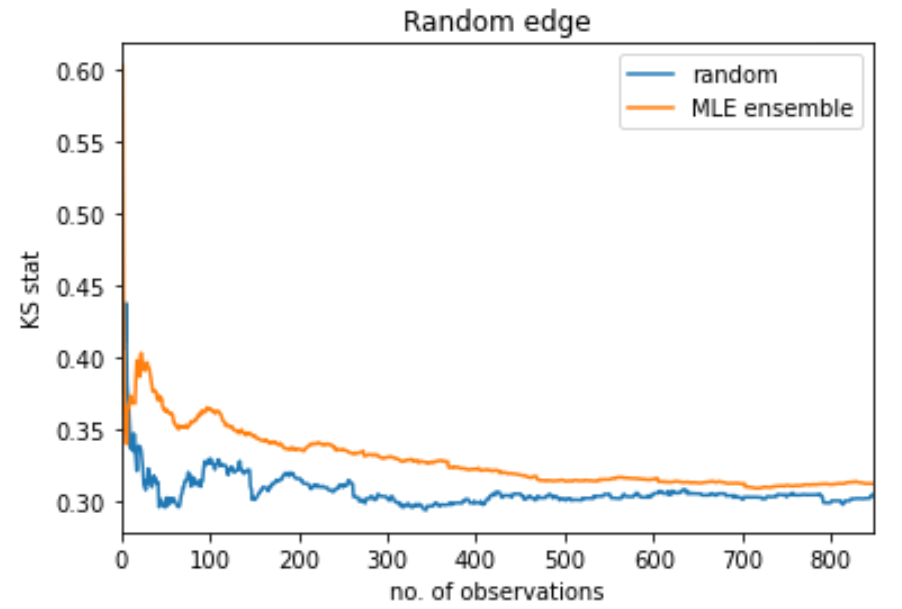
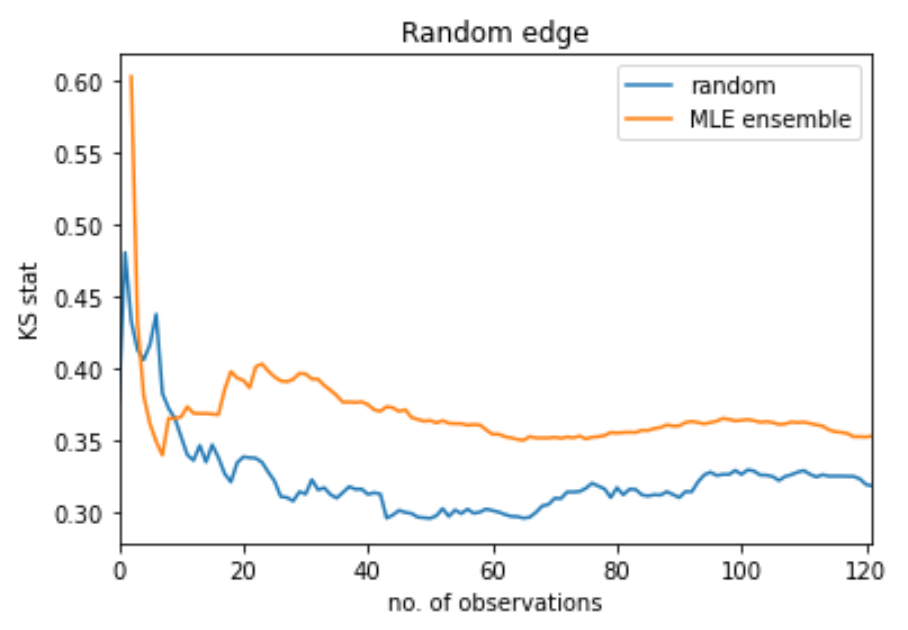
* Train prior on a random edge selected from a different map
* If we train the prior using all the data, then we have a very strong prior belief, which is hard to overcome with new data: see below



* 2 ways to overcome this bias:
  + Allow each datapoint from new map to count more than once
  + Use a small number of datapoints to train the prior
* Comparing using 10, 100, 1000 datapoints to train the prior: when the “wrong” edge is selected as prior, it can take a long time to correct this incorrect prior belief



* However, sometime, even random edge with high prior belief (1000 observations) works well

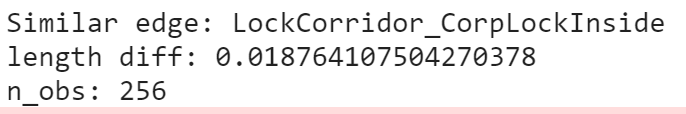


## Edge with most similar length

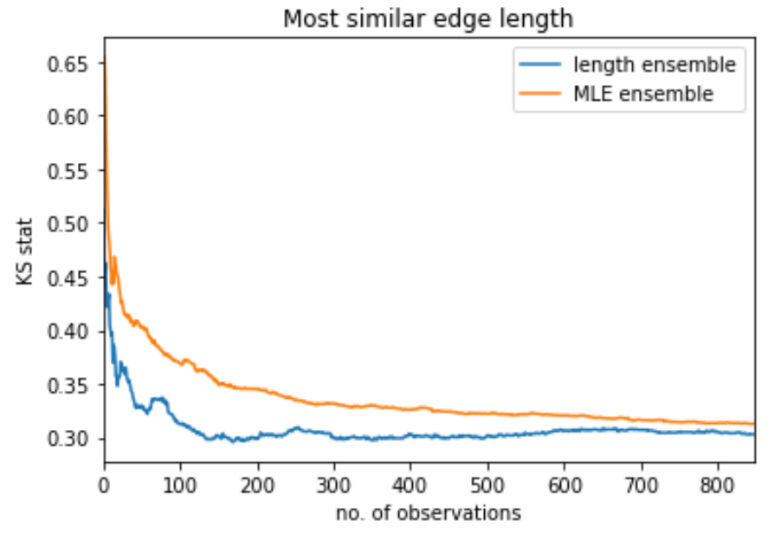
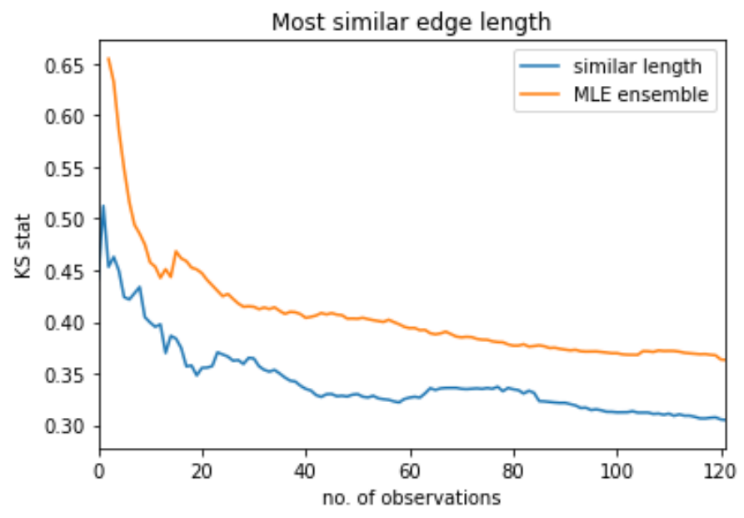
Procedure:

1. Find *n\_similar* most similar edges
2. Choose most similar edge with more than *cutoff* datapoints. Else choose most similar edge

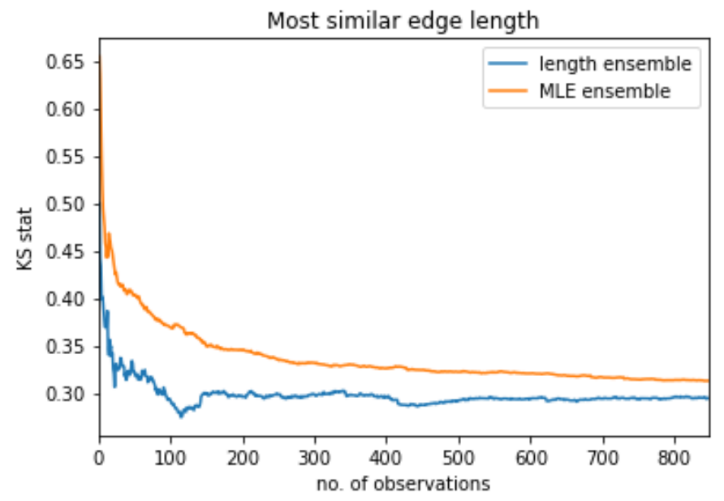
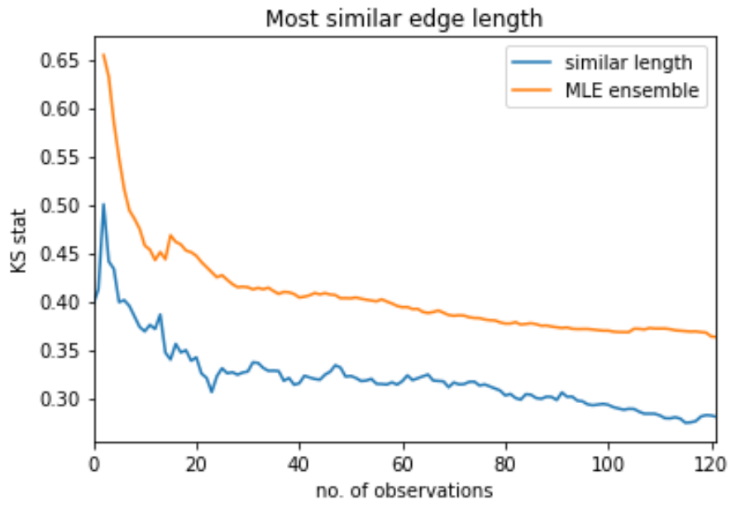
For “WP49\_WP50” in AAF, the most similar edge with more than cutoff = 100 datapoints is



* 10 observations in creating prior. Ensemble method (n\_repeats = 10)

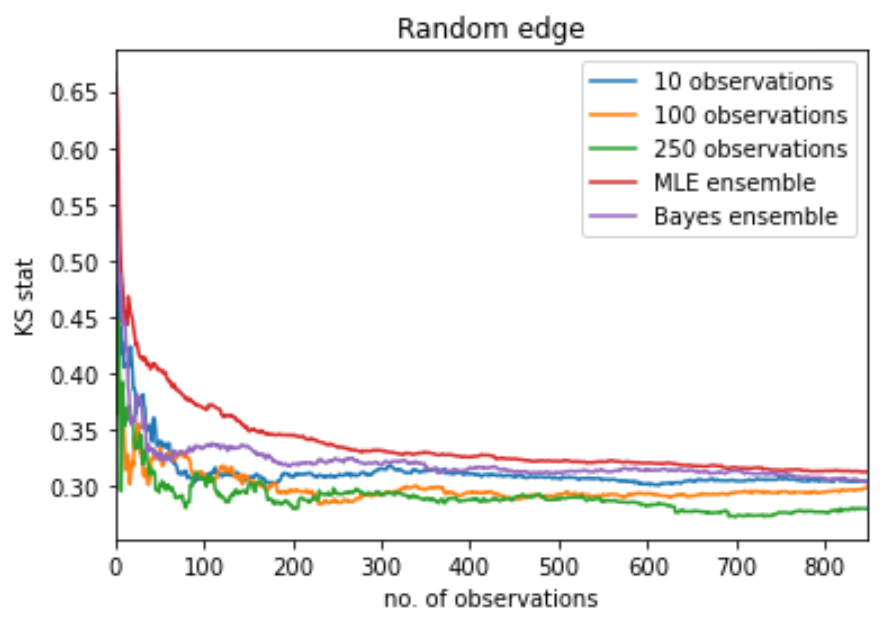
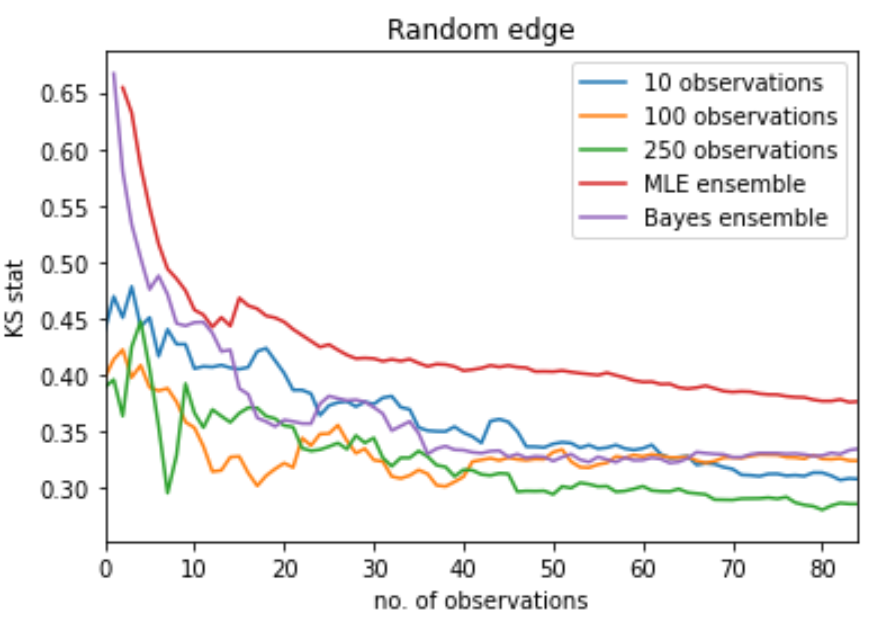


* 100 observations in creating prior. Ensemble method (n\_repeats = 10)

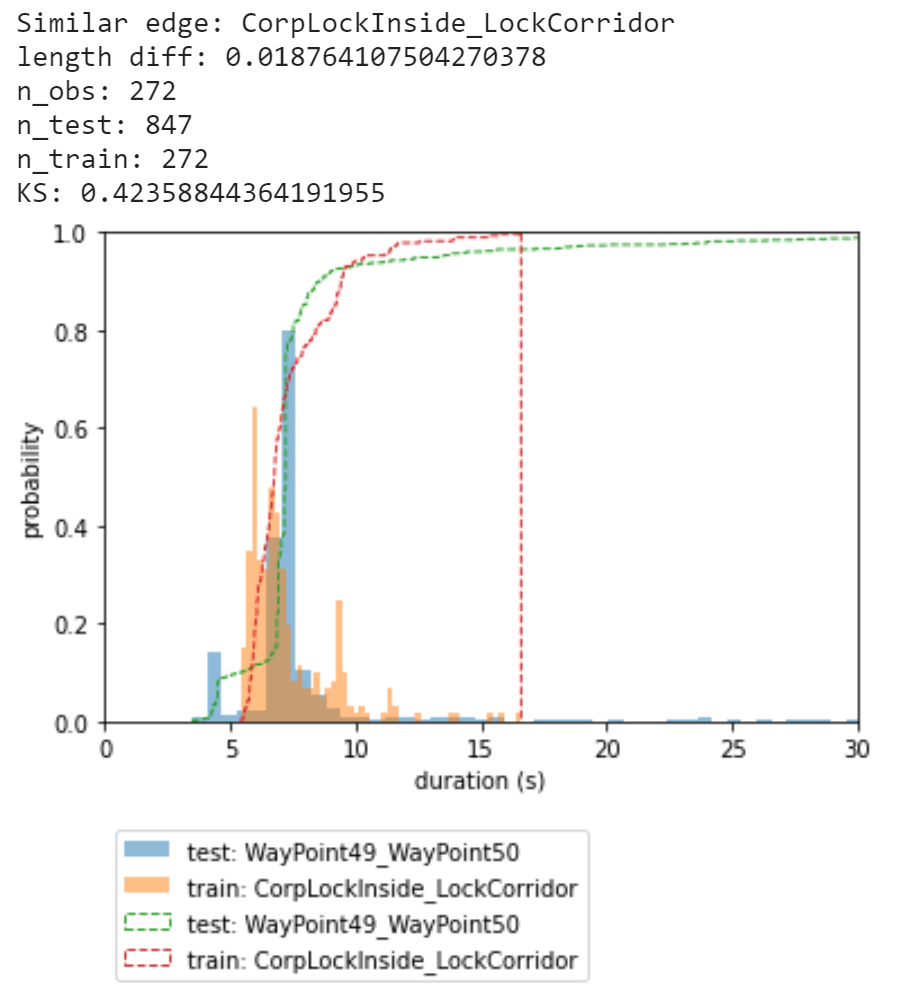


Comparing number of observations in creating prior (with ensemble method – n\_repeats = 10)

* Using more data in prior gives better starting KS
* All values converge



As a sense check, we compare the empirical pdf, cdfs & KS of the test edge vs chosen training edge: good overlap & as shown above, we start with KS at 0.4-0.45 at 0 iterations, instead of KS of 0.65 at 1 iteration.

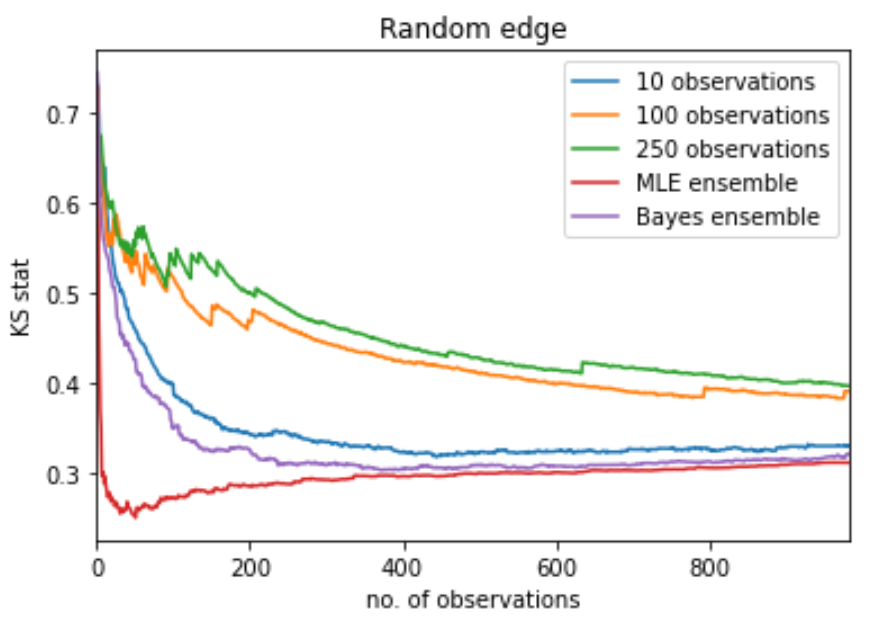
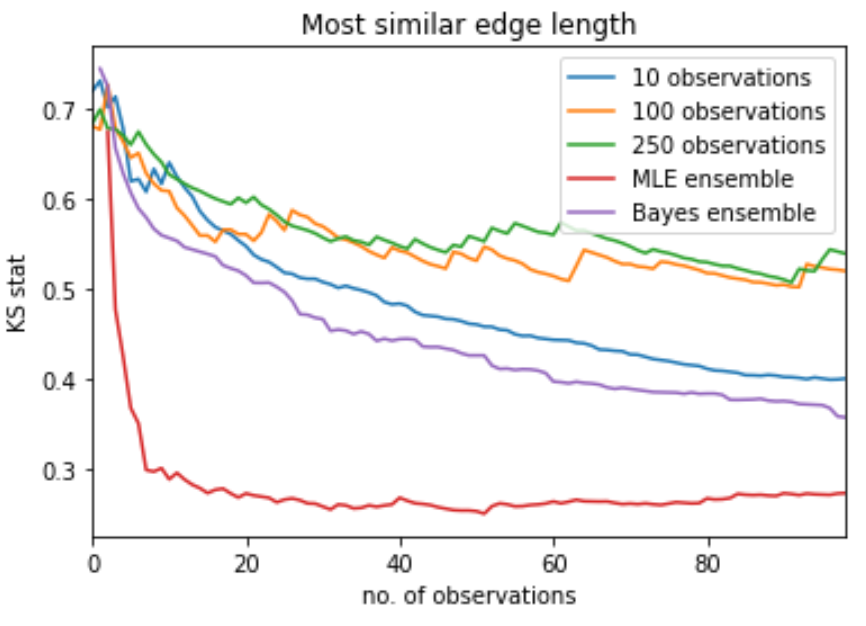


#### An example of length ensemble not working well

**Test on:** “WP\_87\_WP89” in AAF, n\_obs = 981

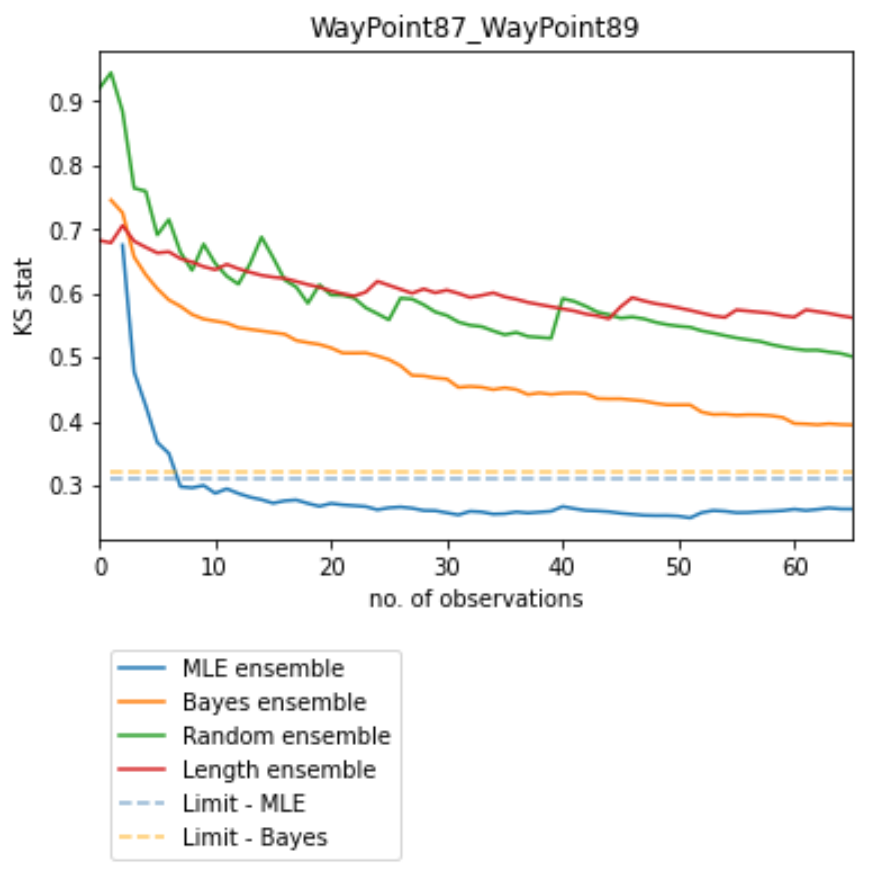
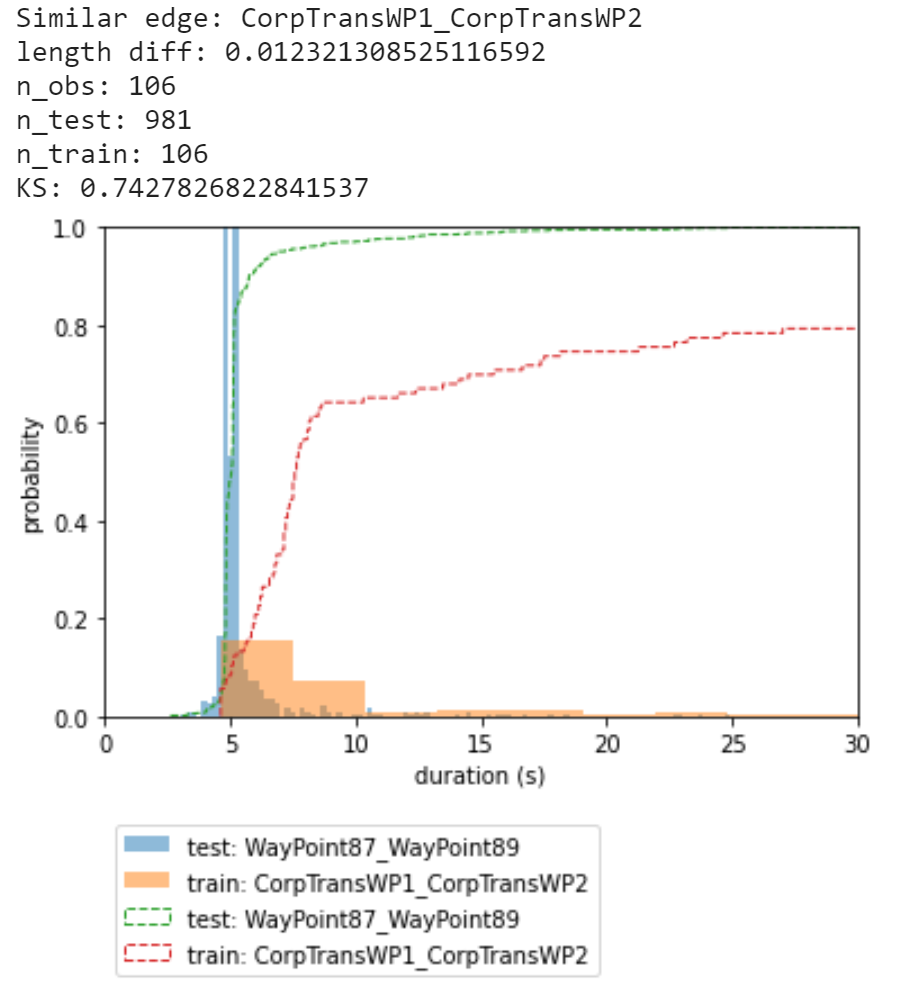
**Train on:** “CorpTransWP1\_CorpTransWP2” in TSC, n\_obs = 106

Edge length diff = 0.012 m



* Any number of data points performs badly. The more data points, the worse the performance is

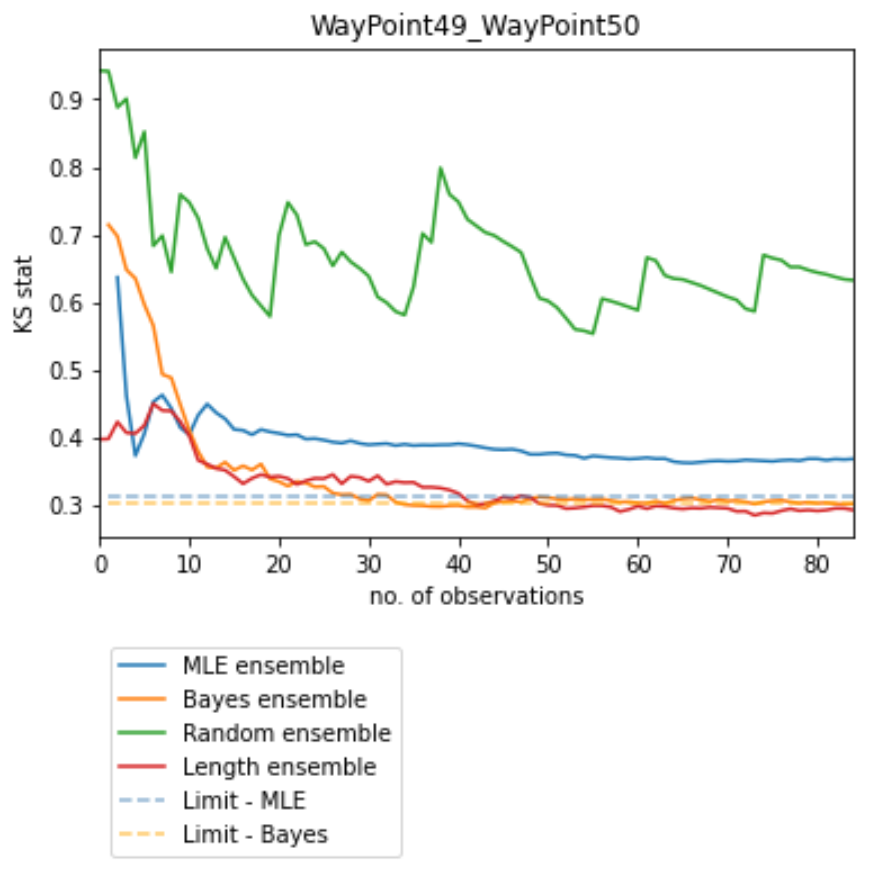
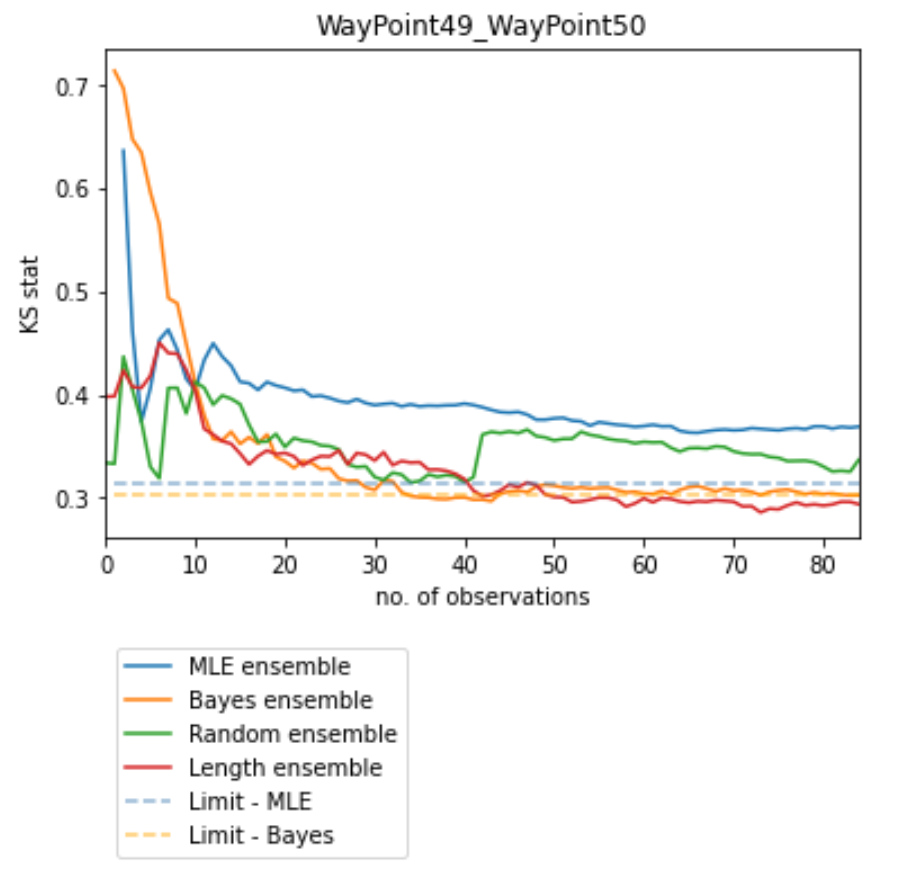
This is not surprising, since the training edge we have chosen is not similar to the test edge. Overall performance:



## Summary of today:

Comparing all methods so far…

* Random ensemble sometimes outperforms MLE & Bayes (naïve) if we choose a similar edge. However, it can also perform very badly
* Length ensemble usually has better short term behaviour & long-term KS (since we start having “seen” correct datapoints and end up seeing more correct datapoints in total)



As a word of caution, it may be good to still choose a moderate number of data points (e.g 50-100) in practice for the methods where we choose an informative prior.

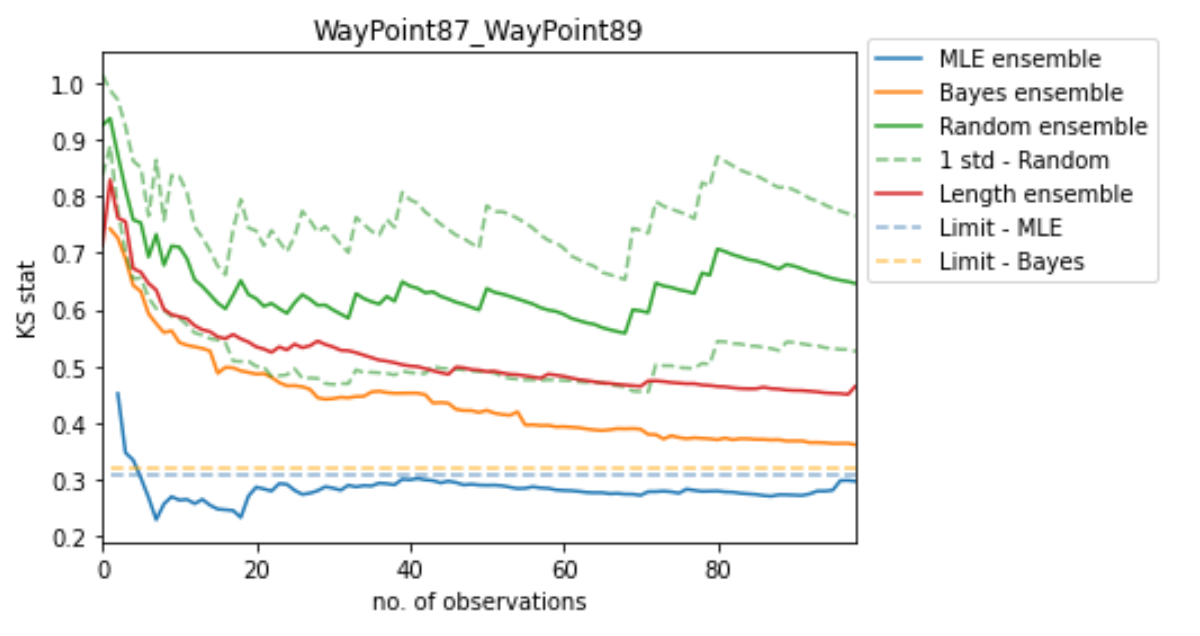
* This is to avoid a mistake that is very difficult to overwrite

It may also be worth making each datapoint that is from the true edge count more than once to help with overwriting any incorrect beliefs.

* This multi-counting of true observations could maybe decay back to 1 as we collect more and more data?

Changing the random ensemble to randomly swap priors on each run

* Calculate variance to reflect uncertainty



# THU: Clustering-Classification

For training on TSC and testing on AAF

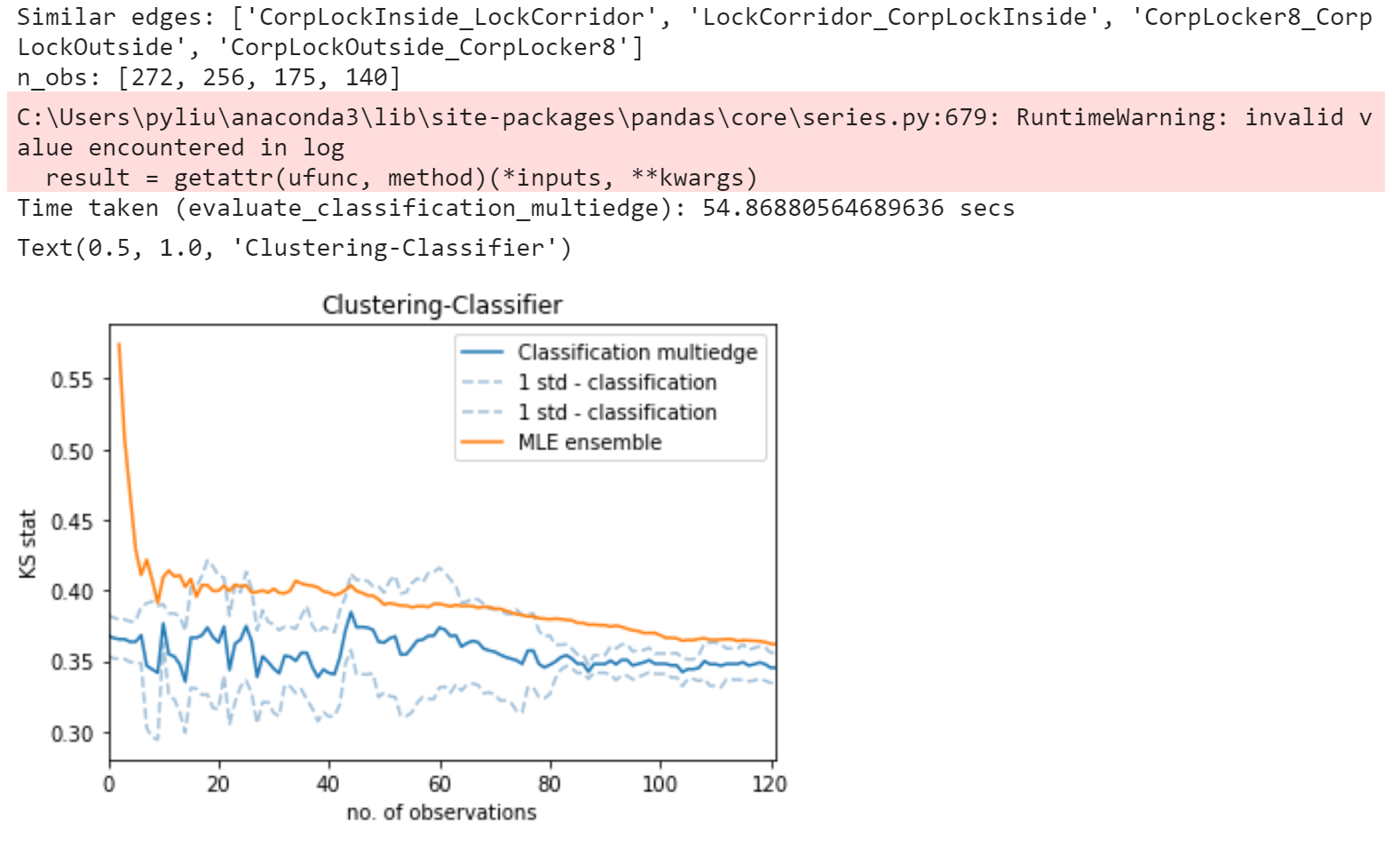
Process:

1. Train clustering on TSC with threshold (already created)
   1. Need to add separate clusters for the edges where we have enough data but that do not fit with the other edges
   2. OR remove those edges
2. Convert to binary class labels & train binary classifier
   1. Random forest?
3. For the AAF edge, compare to all TSC edges and select edges that have highest probability of class 0
4. Train prior using highest probability edge that also has enough data points (>= cutoff)

Ensemble:

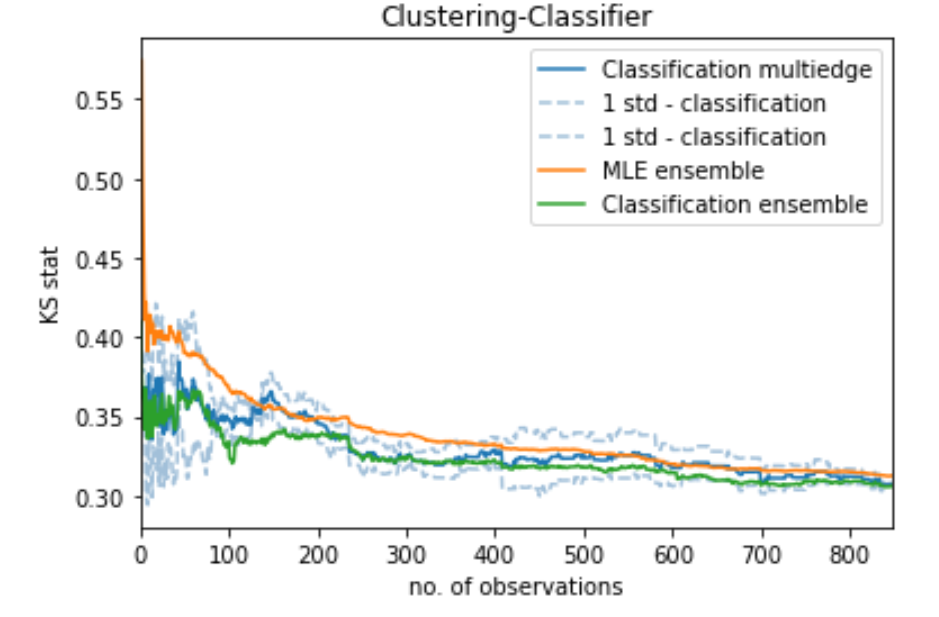


Multiedge:



Comparing Classification Ensemble vs Multiedge:

* Very similar performance
* Ensemble fits inside 1 std dev of Multiedge



## Summary

I made some changes to the types of evaluation methods & added a new prior creation method (Clustering-Classification):

3 evaluation methods:

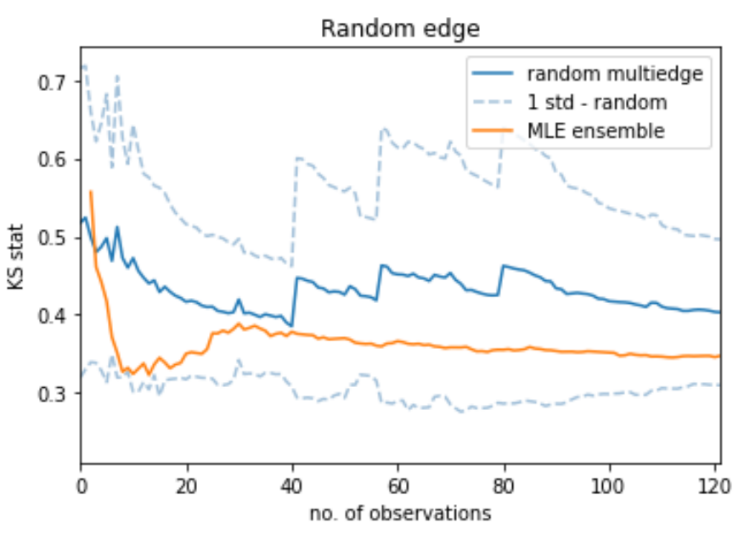
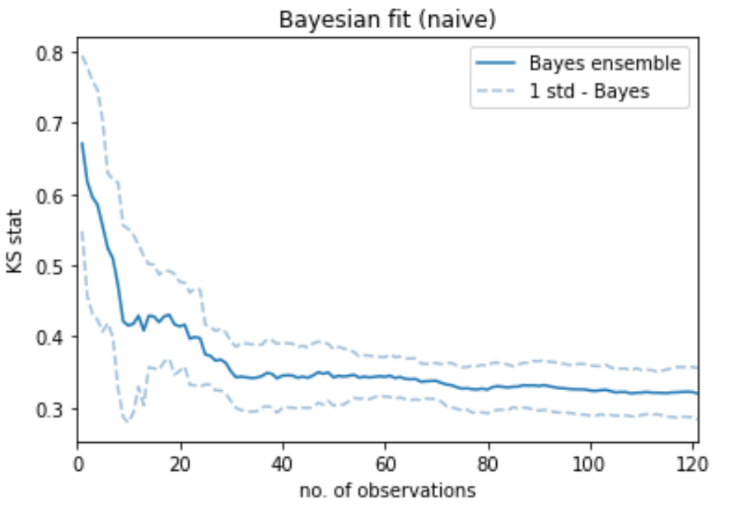
* **Single:**One run on randomly shuffled dataset. Short-term performance is highly dependent on order of shuffled dataset.
* **Ensemble:** Calculate mean and standard deviation of KS for *n\_repeats*runs. Datasets are randomly shuffled before each run. The same edge is used for each run
* **Multiedge:** We have a list of edges for forming the prior. A different edge is used for each run, and datasets are randomly shuffled before each run. Calculate mean and standard deviation of KS for all runs.

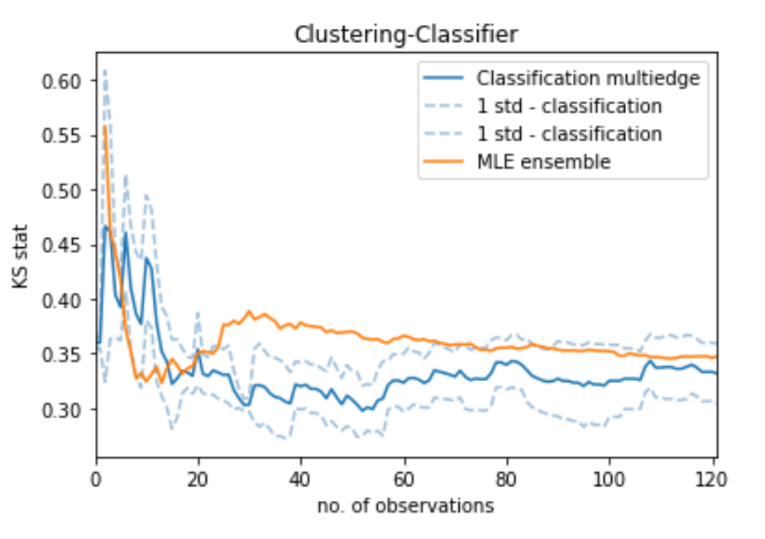
5 prior creation methods:

1. **MLE** - no prior
2. **Bayes**- uninformative prior
3. **Random**- train prior on *max\_obs* datapoints from a random edge with at least *cutoff* datapoints
4. **Length** - train prior on *max\_obs*datapoints from the edge with lowest length difference with at least*cutoff* datapoints
5. **Classification** - train on *max\_obs* datapoints from most similar edge with at least *cutoff*datapoints
6. First step is HAC clustering on training map with thresholding to remove edges that are not similar to any of the other edges in the map
7. Then convert clusters into a binary label. *Class 0* = edges are not in the same cluster. *Class 1* = edges are in the same cluster.
8. Train classifier (random forest/gradient boost) to predict whether 2 edges are similar. Use classifier to find *n\_similar* training edges that have highest probability of *class 1*(same cluster) compared to the specified test edge (from test map)
9. Train prior on datapoints from most similar edge (highest class 1 probability), with at least *cutoff*datapoints

In terms of the standard deviation:

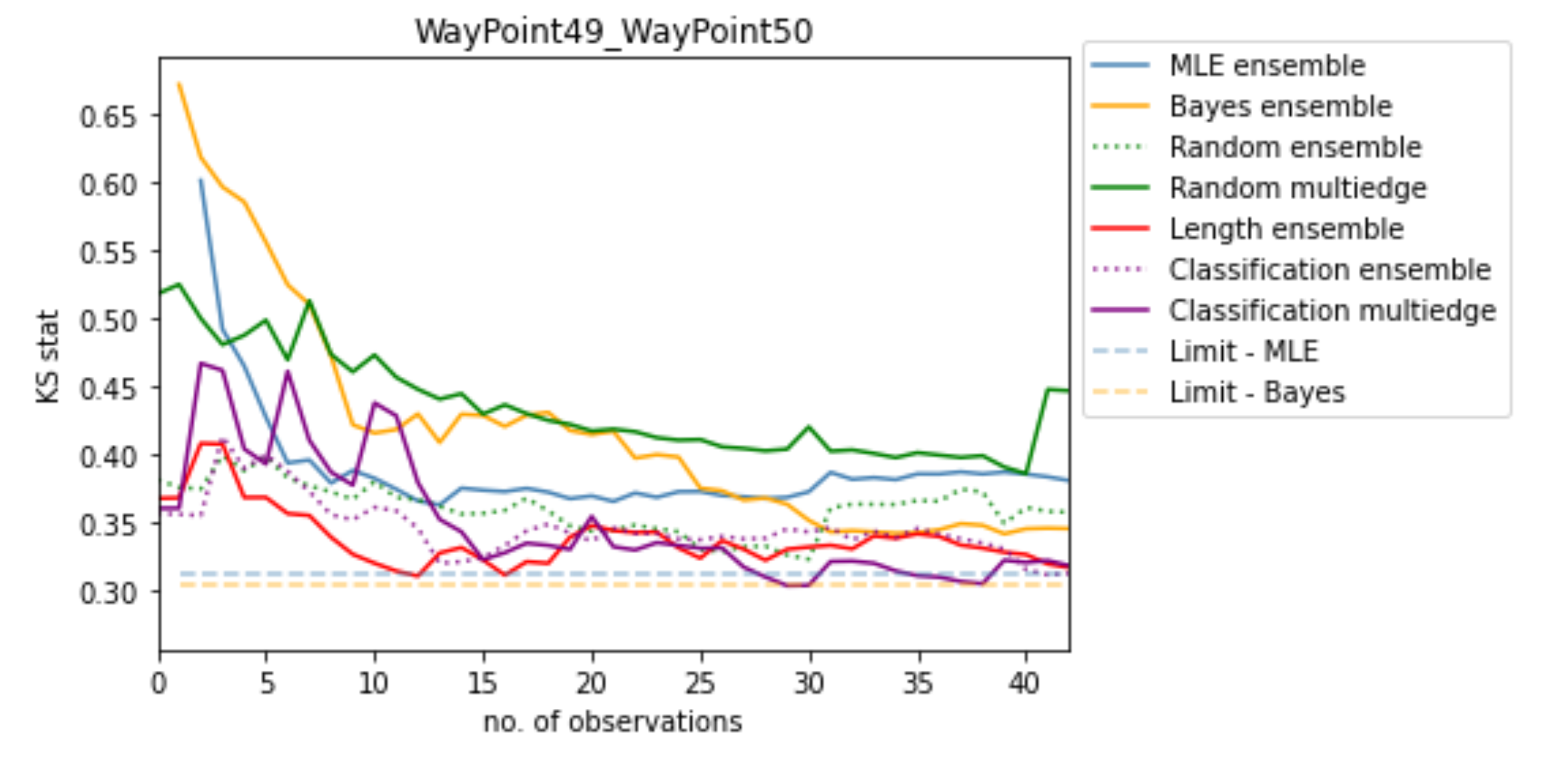
* Using 10 runs for ensemble methods, there is a fairly low standard deviation (attachment 1)
* With **Random Multiedge** (attachment 2), we have very large standard deviation for 10 different edges), even in the long-term. This suggests that the random edges are not similar (obvious)
* With **Classification Multiedge** (attachment 3), we also have low standard deviation for 4 similar edges. This suggests that the chosen edges are fairly similar





As a comparison of methods so far (graph below):

* **Random multiedge** prior has a mean KS that converges very slowly to the optimum KS value. If the true edge does not have enough datapoints, it will not reach this optimum.
* **Length** & **Classification** priors can offer useful information in the short term, to offer better performance than the uninformed **MLE**& **Bayes** methods
* For this edge in particular, the most similar edge with >100 datapoints is the same for both **Length Ensemble** & **Classification Ensemble** methods (hence very similar line).
* **Classification Multiedge** also uses data from a few other edges, but the performance is not significantly worse than **Length Ensemble** or **Classification Ensemble**, especially the starting KS at *no. of observations* = 0

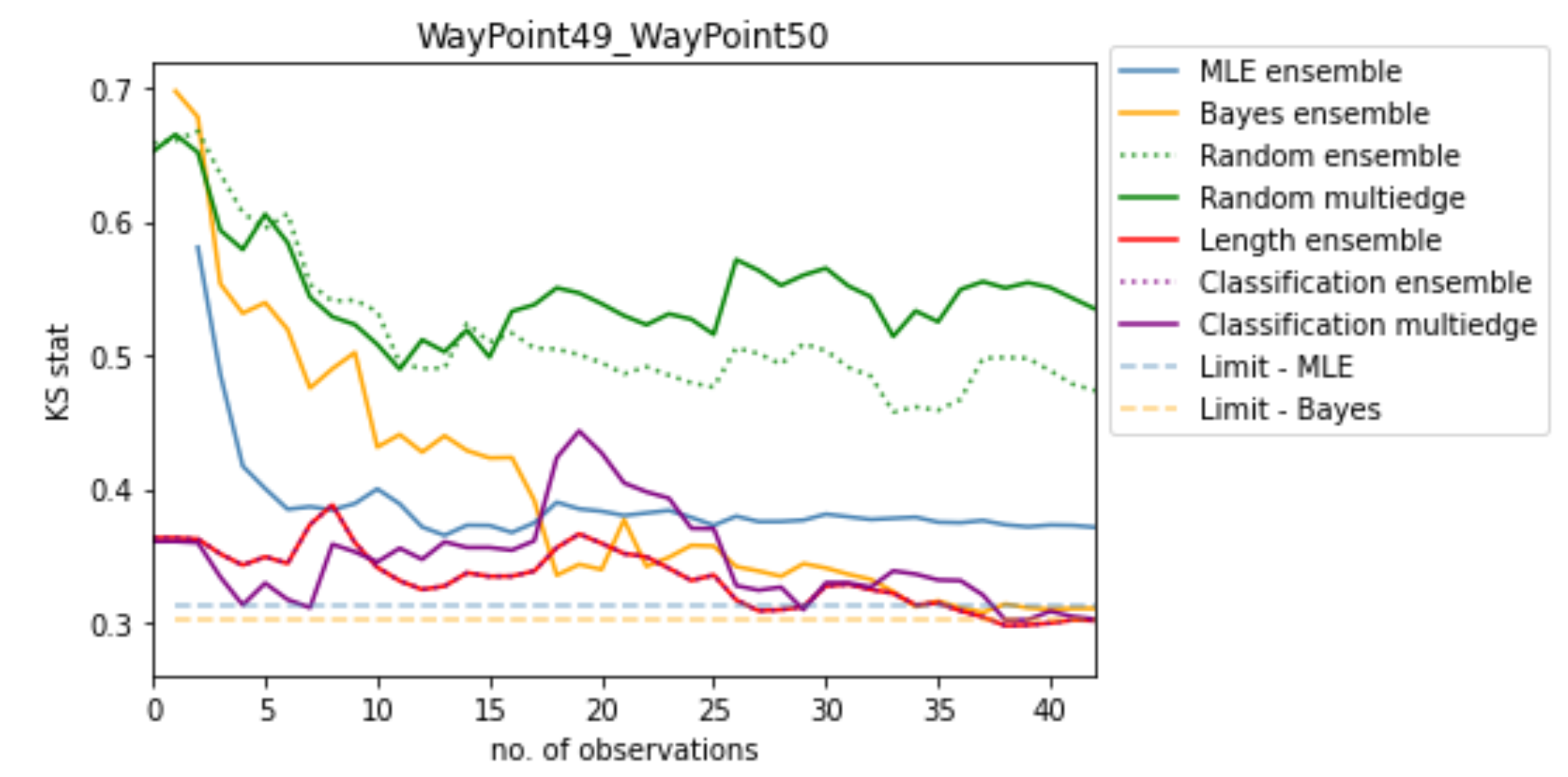


Next steps:

1. Implement KS Regression-based prior creation method
2. Use a random\_state for the dataset shuffling for reproducibility?
   1. e.g. generate a random list of integers of length n\_repeats (this uses a fixed random seed = 1). Then use this random list as the seeds for shuffling
   2. Or create a list [1,2,3... n\_repeats] and use that as the seed for shuffling
   3. This could make the comparison between different prior creation methods more direct sense they use the same shuffling of data
3. Look at a wider variety of edges to see where the Classification & Regression methods outperform/underperform the Length method
   1. Does length still work when congestion is present?
      1. For the Blenheim map, all the edges are the same length, but with higher levels of congestion, it would doesn't make sense that the distributions would be the same
   2. Edge towards lift in AAF
4. Use fewer features in classifier/regressor?

# FRI: KS Regression

## Use seed to reproduce results between prior creation methods



Process:

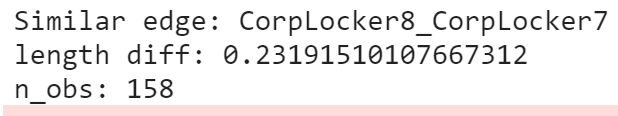
1. Input a *random\_state* seed
2. This generates a list of random integers of length *n\_repeats* or *len(valid\_edges)* for Classification method
   1. If we use the same *random\_state* seed, the list of random integers is the same for each evaluation method
3. These random integers are the random seeds for the shuffling that occurs before each run.

This means that by using the same *n\_repeats* & *random\_state*, all evaluations see the same data in the same order.

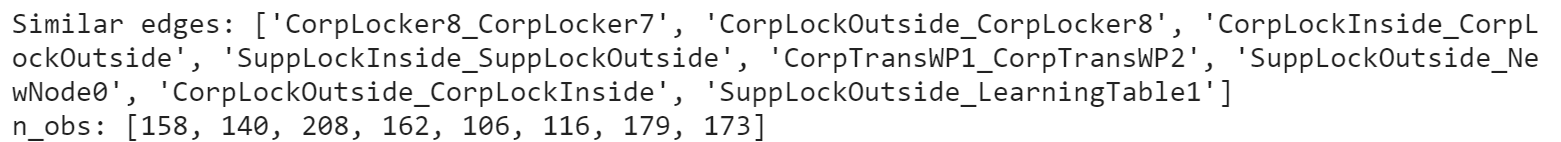
## KS Regression

Regression predicts a different edge to Length & Classification

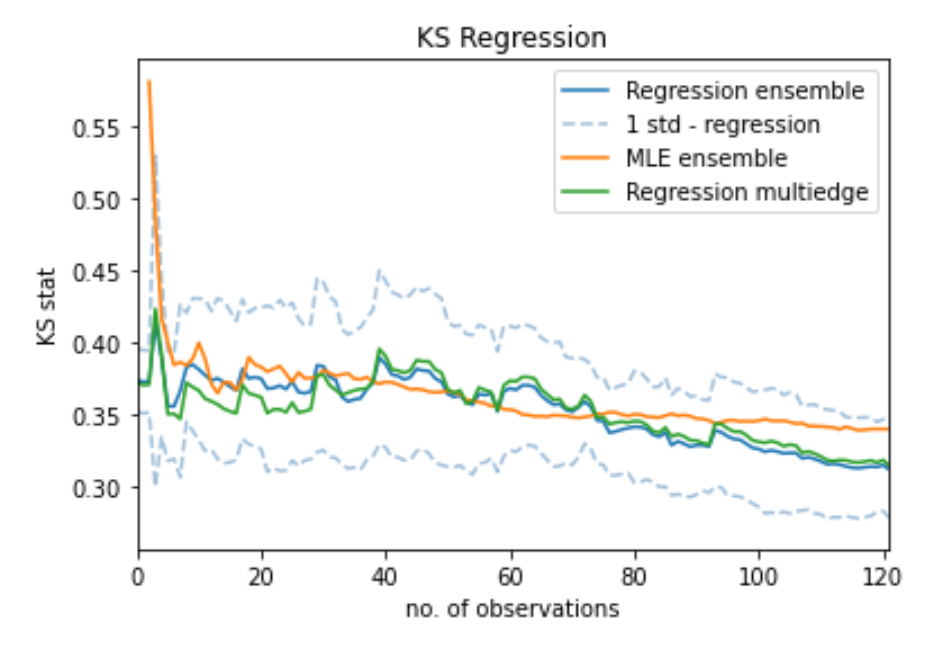
For WP49\_50 (ensemble):



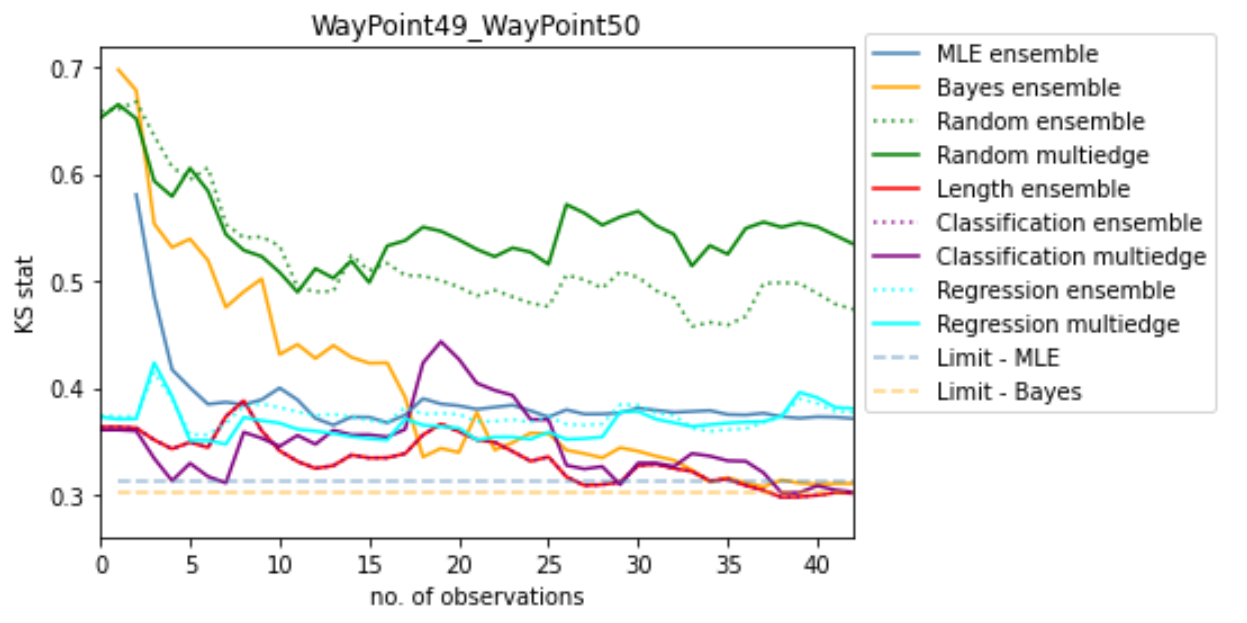
For WP49\_50 (multiedge):



In both Regression ensemble & Regression multiedge, we have better short-term & long-term performance than MLE. Ensemble & Multiedge have similar performance vs each other



## Summary



# Next steps – look at more edges:

1. Problematic edges in the clustering
2. Edges with high levels of congestion, but similar length (Blenheim might be good for this)
3. A holistic overview of all edges in a map: plot a bar chart of mean ± std of the starting KS for each method
4. Documentation: