**Week 7 Writeup**

# RECAP: Week 6 Summary

1. **[MON]** – update congestion algorithm, clustering in Blenheim map (for different congestion levels)
   1. Use the vector/scalar context files for the clustering
   2. Test against the data in the files
   3. Try using date started & date finished – use pandas function between
2. **[TUE] –** generalise between edges (2 methods)
   1. **Crps\_between\_edges.py:** fit distribution on one edge, test against data from another edge
   2. **Mergefit.py:** fit distribution on N datapoints from each edge
      1. **Try it out randomly**
      2. **Use edges with only a little data for comparison.**
      3. **Try using a proportion from all edges e.g. 0.5**
   3. Look online: are there issues with CRPS. Does the CRPS favour the mean?
   4. Does the min change?
   5. Use CVM / KS
   6. How to interpret scale of CVM
3. **[WED]** – Fixed parameterisation discrepancies between Scipy and my implementation. Implement **mergefit\_crps\_ks.py** with plots/CRPS/KS values
4. **[THU]** – Linear regression for lognormal parameters vs edge length
   1. Offset is clearly linear
   2. Mean could be linear. Var could be polynomial. Probably influenced by other factors
   3. To do: try multivariate regression
5. **[FRI]** – Fix **mergefit\_crps\_ks.py**
   1. KS is a better metric due to intuition:
      1. Original fit is better than mergefit according to KS. Vice versa for CRPS
      2. KS is bounded between 0 & 1 so it is clear what constitutes a “good” KS score
   2. Around 5 datapoints is the best n\_samples
   3. Draw AAF, TSC, LABS maps – AAF or TSC might be most appropriate. I might choose AAF for the simpler names + it’s clear that some edges “should” be in the same cluster

# PLANS: Week 7

1. **[MON]** - Fix issues from last week (check congestion algorithm against scalar context, randomise datapoints for mergefit, answer questions about CRPS/KS/CVM, fix my implementation of CVM test)
2. **[TUE]** - Fit distributions for STRANDS datasets. Is lognormal still a good fit?
3. **[WED]** - Cluster edges according to KS, CVM, edge length, connections, max angle of turning etc.
4. **[THU]** - Classification based on factors (edge length, connections, etc). Output is binary: do 2 edges belong to the same cluster?
   1. Input = factors, Output = Similarity measure for a regression
   2. Input = factors, Output = Binary classification for whether 2 edges are similar
      1. Build a ground truth using KS/CVM
5. **[FRI]** - Test classification on the other map against a "ground truth" determined by KS/CVM clustering

**[OTHER]** – Plot parameters vs factors. Any correlations? Any other factors

**[OTHER]** – Multivariate regression for parameters vs factors – cautious

* 1. output = lognormal params
  2. input = factors that affect clustering

# Mon: Fix Issues from Wk6

#### Congestion

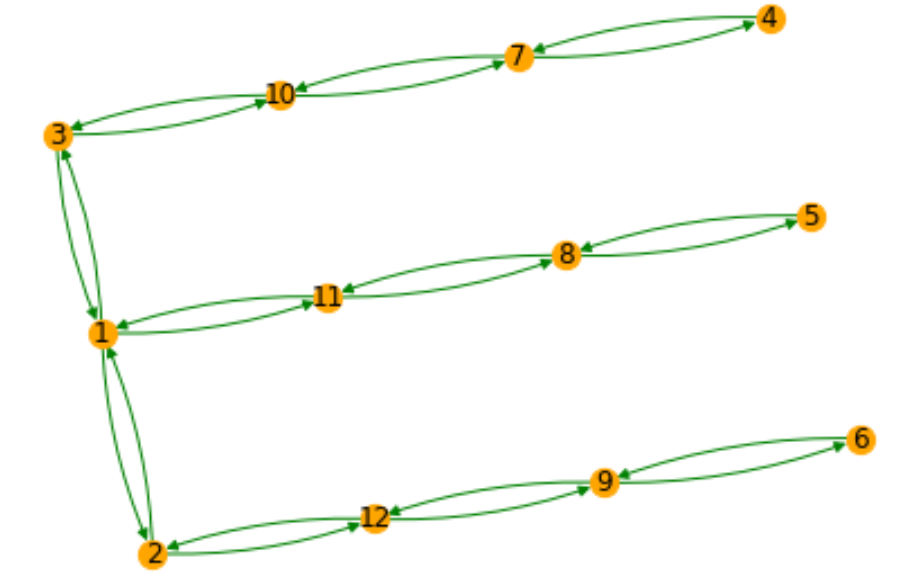
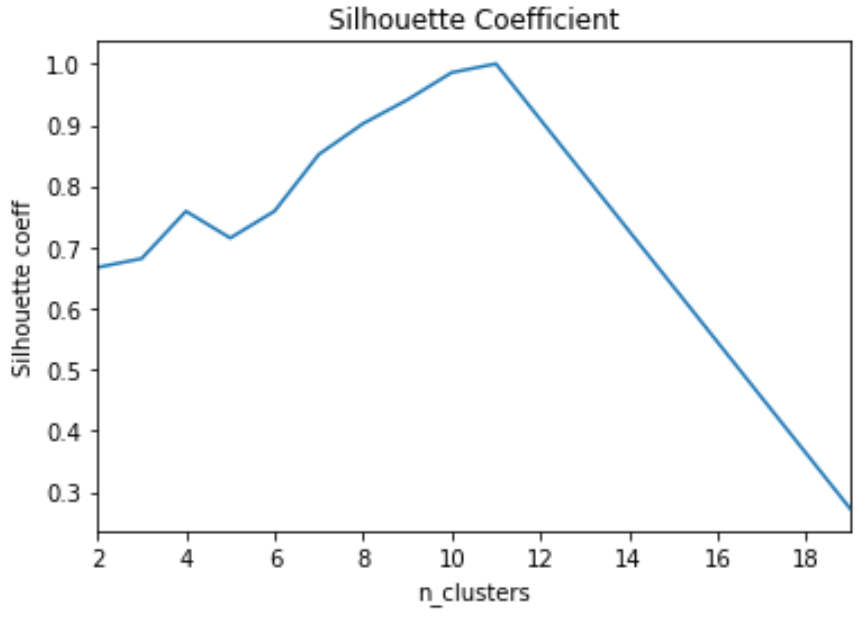
Used data from YAML files (processed with scalar contexts).

Function to get data: **get\_data\_yaml.py**

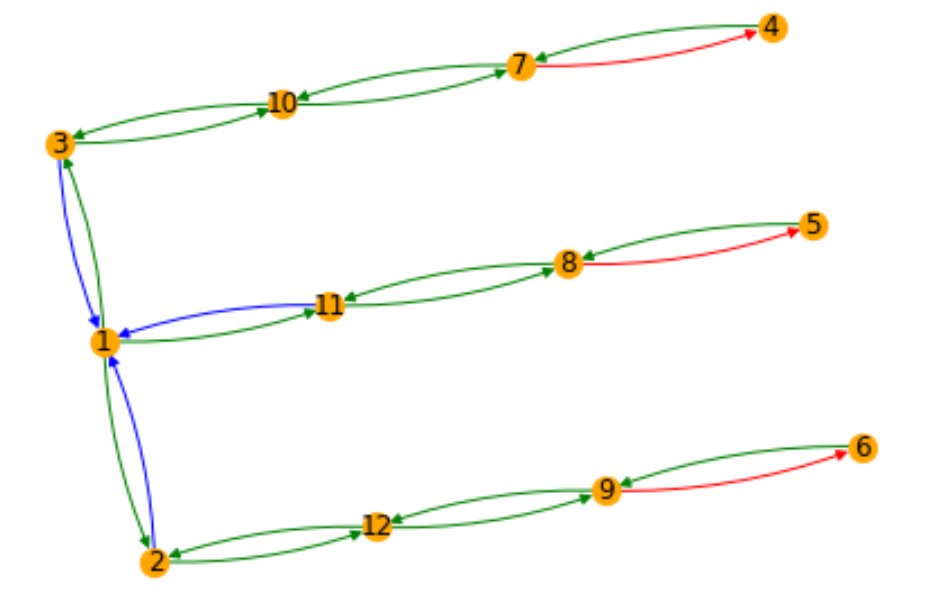
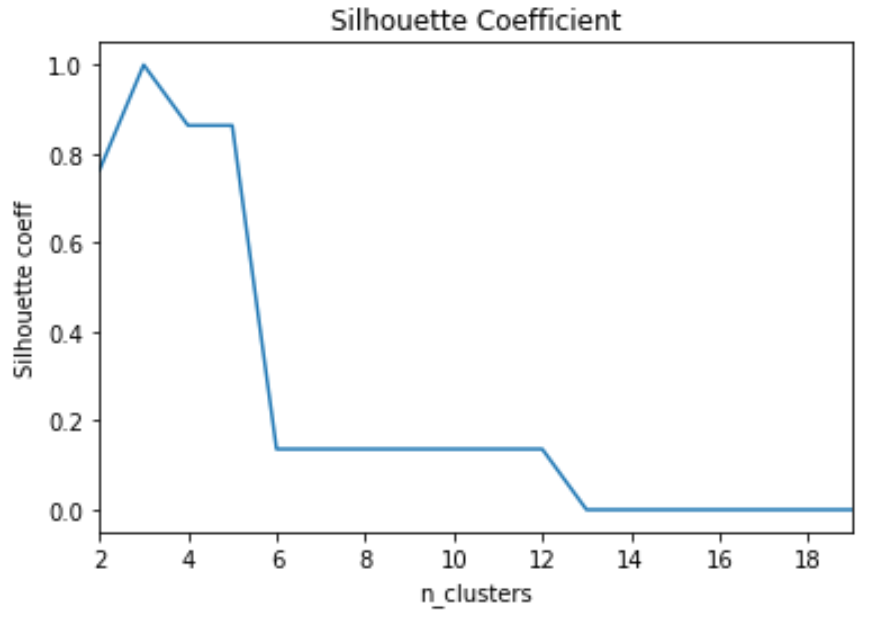
Execute clustering algorithms in **MAIN7.ipynb**

**Clustering by spatial feature:**

Cluster by edge length:



Cluster by connections:



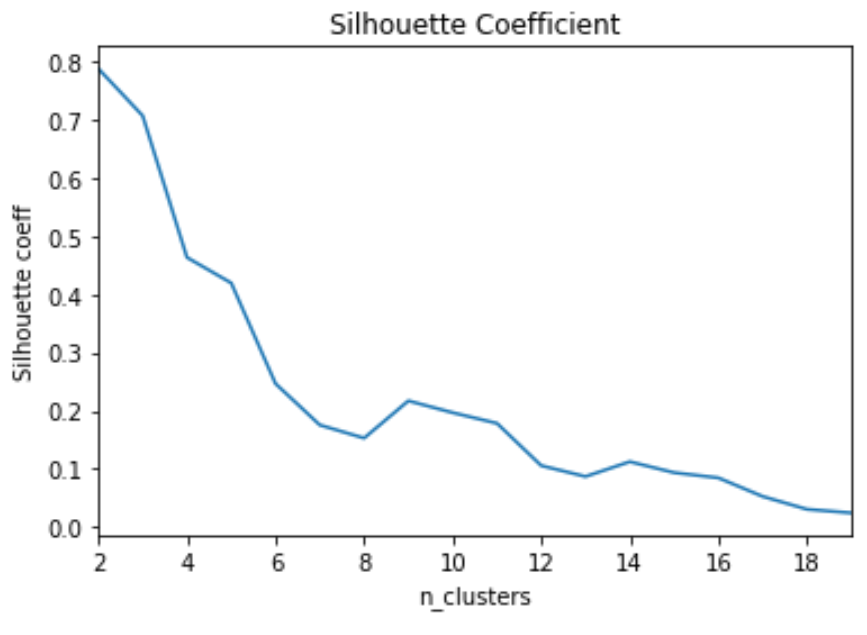
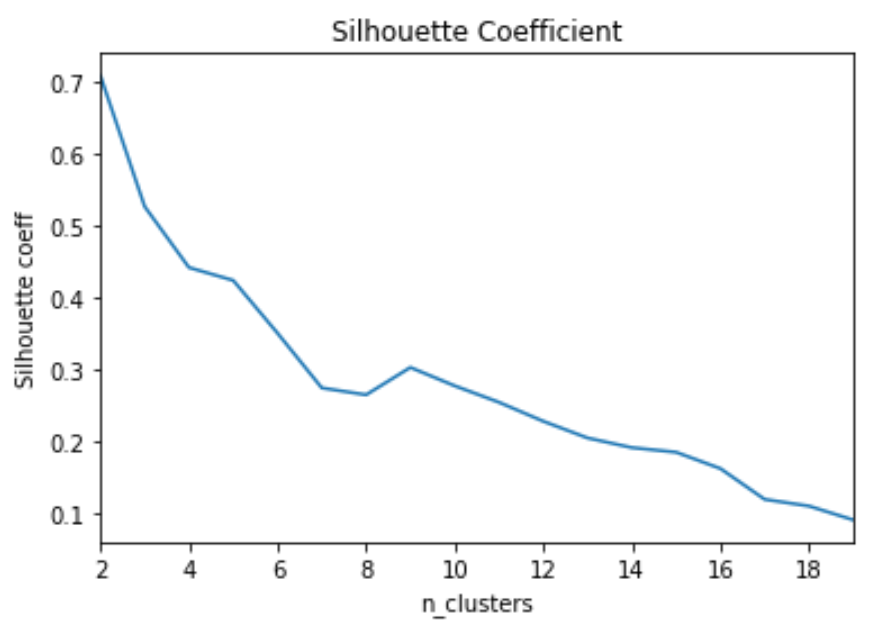
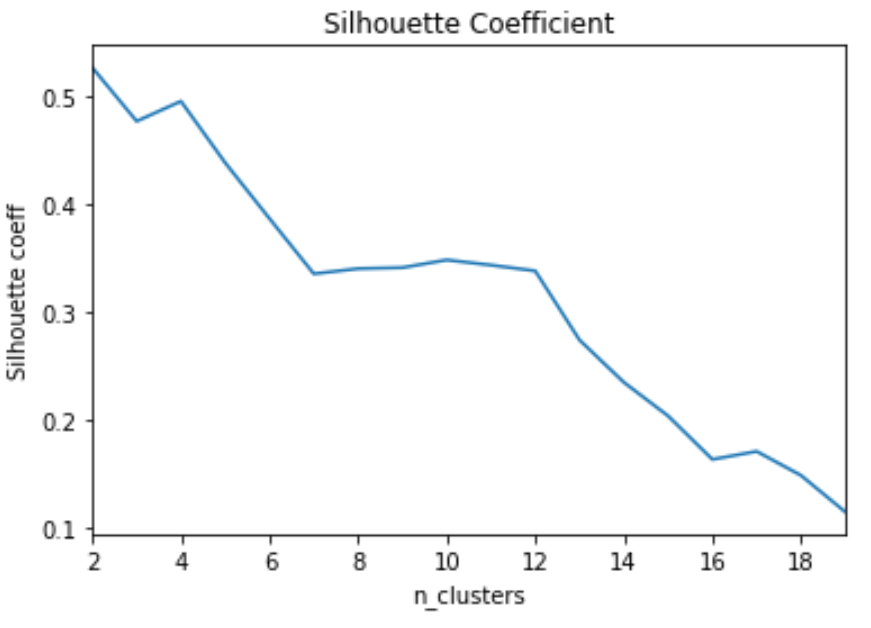
**Clustering by KS metric for different levels of congestion**

1 cluster for all levels of congestion

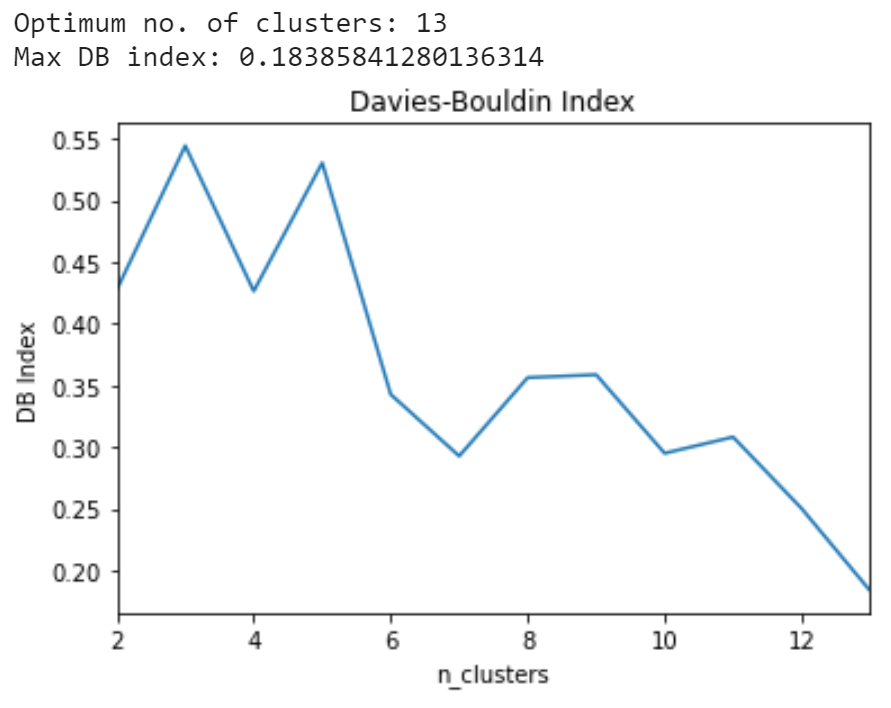
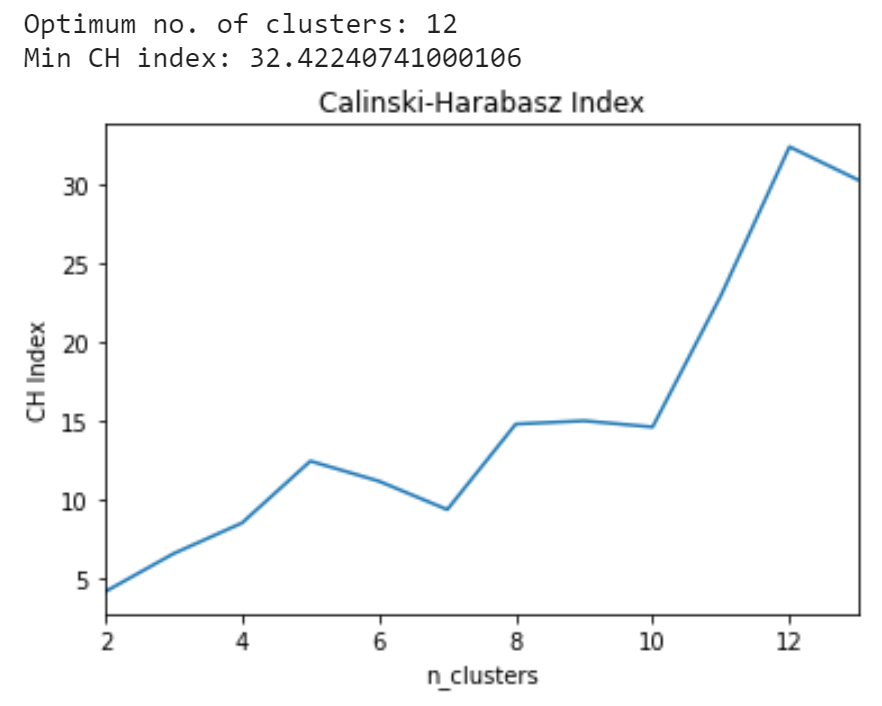
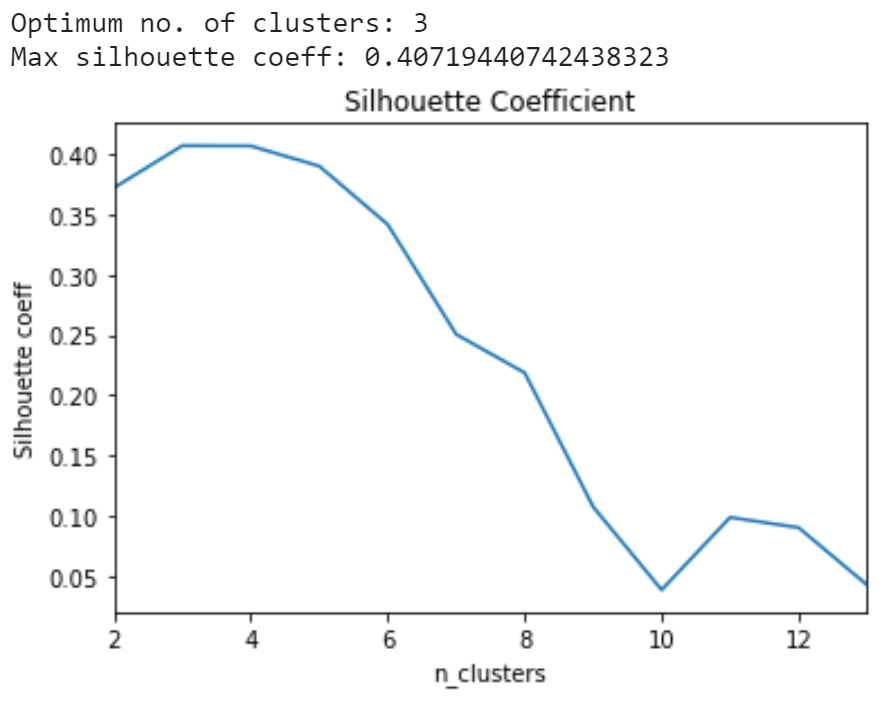
We have:

* 12995 datapoints for 1 robot
* 7818 datapoints for 2 robots
* 1924 datapoints for 3 robots
* 216 datapoints for 4 robots
* 9 datapoints for 5 robots

1 robot 2 robots 3 robots



4 robots



#### Randomise data for mergefit

Use pandas sample function:

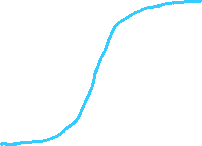
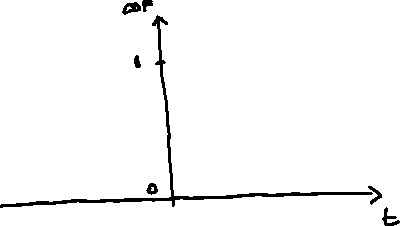
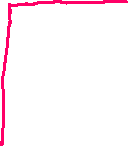
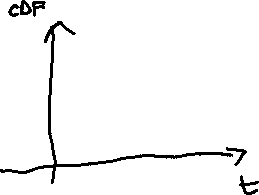
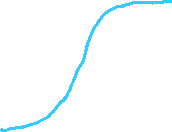
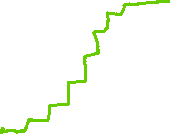
**df = df.sample(frac = 1).reset\_index(drop = True)**

There is little difference between successive randomisations (see **MAIN6.ipynb**), but this is a necessary step to allow for “generalisation” arguments.

#### Clarify CRPS / KS / CVM

CRPS is like one sample CVM. Taking median/mean is not an efficient use of available data.

* In practice, taking the mean favours smaller samples, since we are less likely to get extreme values.
* Taking the median artificially creates high CRPS scores since a score close to the peak is the lowest possible
* CRPS is very useful if the distribution is unknown or can not be easily specified. However, in our situation, we do know the distribution (lognormal) so CRPS provides no advantage over CVM



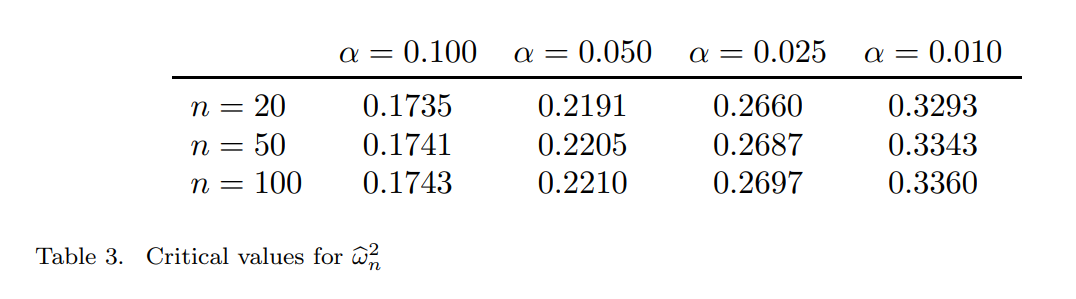
Red is single sample EDF for CRPS

Blue is CDF of predicted distribution

Integral of the square distances between curves is CRPS

TLDR: CRPS is not a great metric if you have lots of new observations

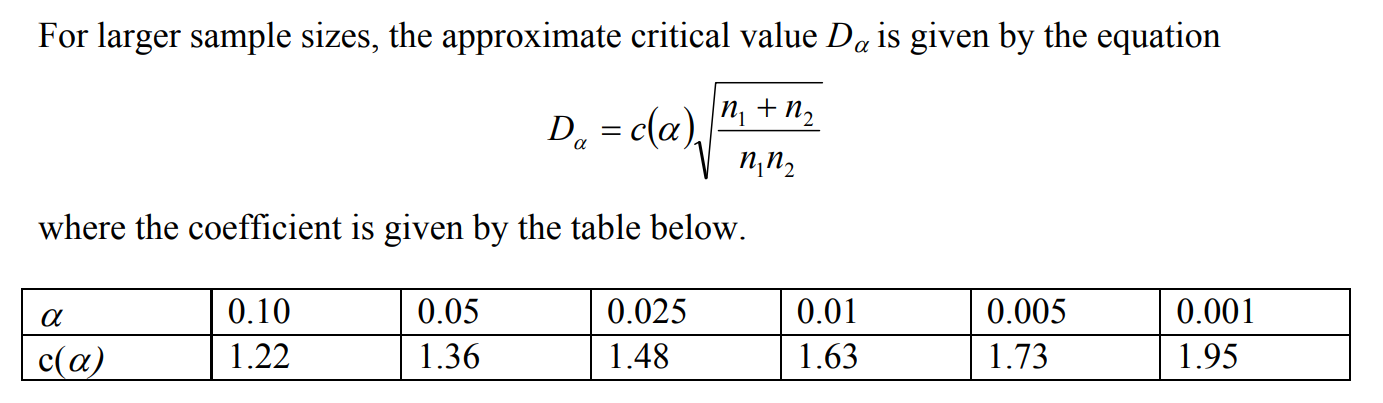
**Critical values for CVM (Baringhaus\_CVM): i.e. we want CVM score less than 0.22**



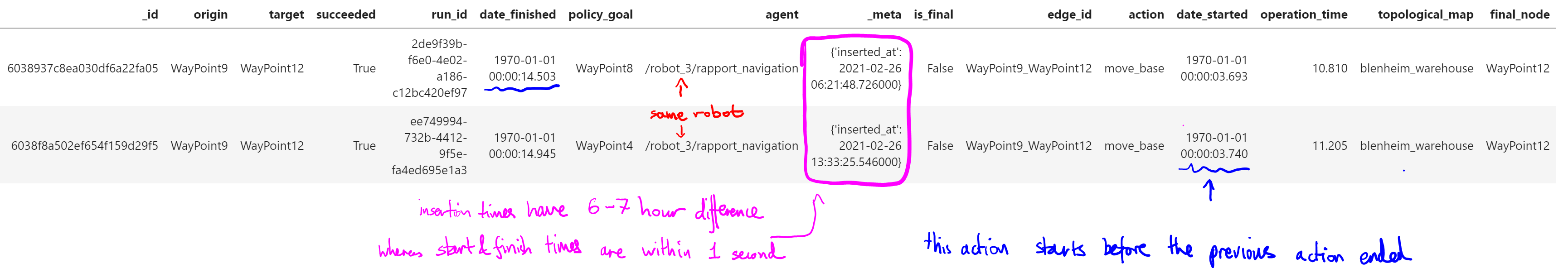
Critical values for 2-sample KS (**Massa\_KS**):

**D\_crit = c(alpha) / SQRT(n)**

Critical values for 2-sample KS (**PWessel2014\_CriticalKS**):



#### Fix implementation of congestion algorithm



**Clarify what time to use:**

For the same robot:

1. Robot 3 has overlapping “date\_started” & “date\_finished” with a different run completed by Robot 3, suggesting that the robot timer may have been reset?
2. However, the “\_meta.inserted\_at” times are different, suggesting this is the actual time, which was not reset

This is not unique to these 2 datapoints. For Robot 3, it has many entries of “date\_started” within 2 seconds:

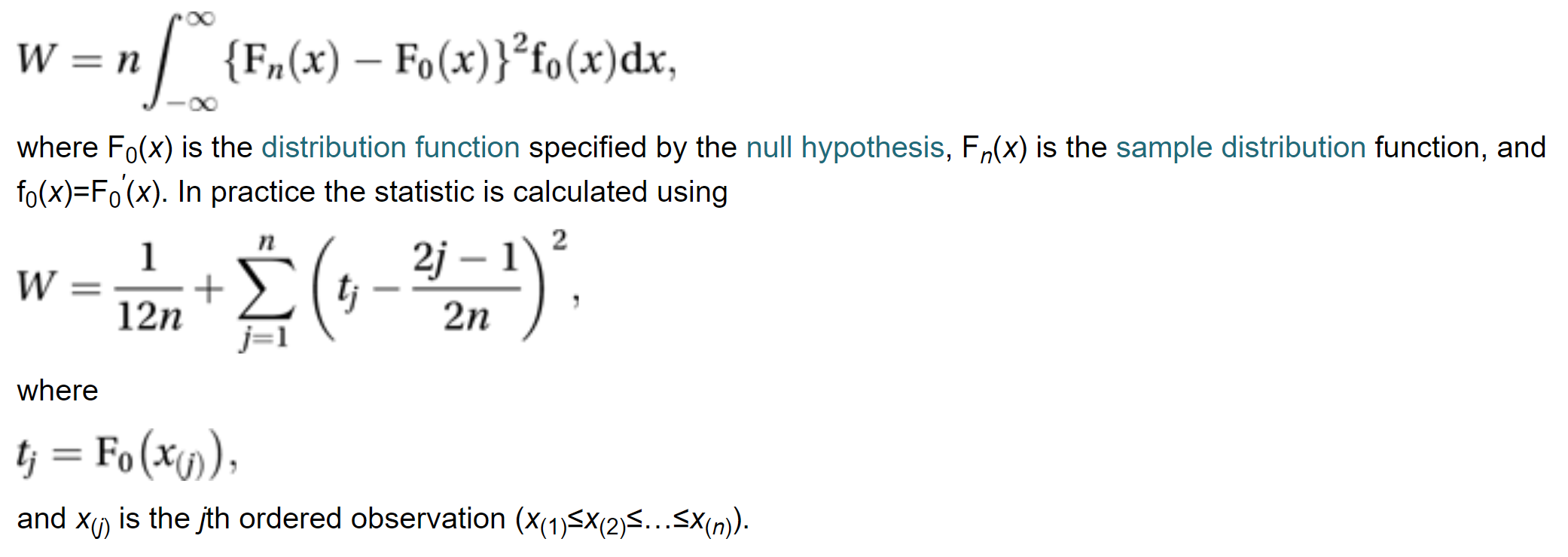


**Should we be using “\_meta.inserted\_at” as the time when we consider congestion?**

#### Fix implementation of CVM

Source for CVM implementation: [scipy/\_hypotests.py at v1.7.1 · scipy/scipy (github.com)](https://github.com/scipy/scipy/blob/v1.7.1/scipy/stats/_hypotests.py#L266-L379)

My understanding of CVM was previously wrong: (see [Cramér–von Mises test - Oxford Reference](https://www.oxfordreference.com/view/10.1093/acref/9780199541454.001.0001/acref-9780199541454-e-1882))



There are extra factors of **n** (no. of observations) and **f(x)** – the predicted pdf in the integral

My CVM & KS implementations are fixed in the Wk7 folder (**error\_ks.py, error\_cvm.py, error\_ks\_2samples.py**)

I also implemented 2 new metrics:

* **error\_abs.py:** integrates absolute difference between CDFs
* **error\_square.py:** integrates squared difference between CDFs (what I though was CVM before)

# TUE: Fitting all Scipy MLE distributions to AAF

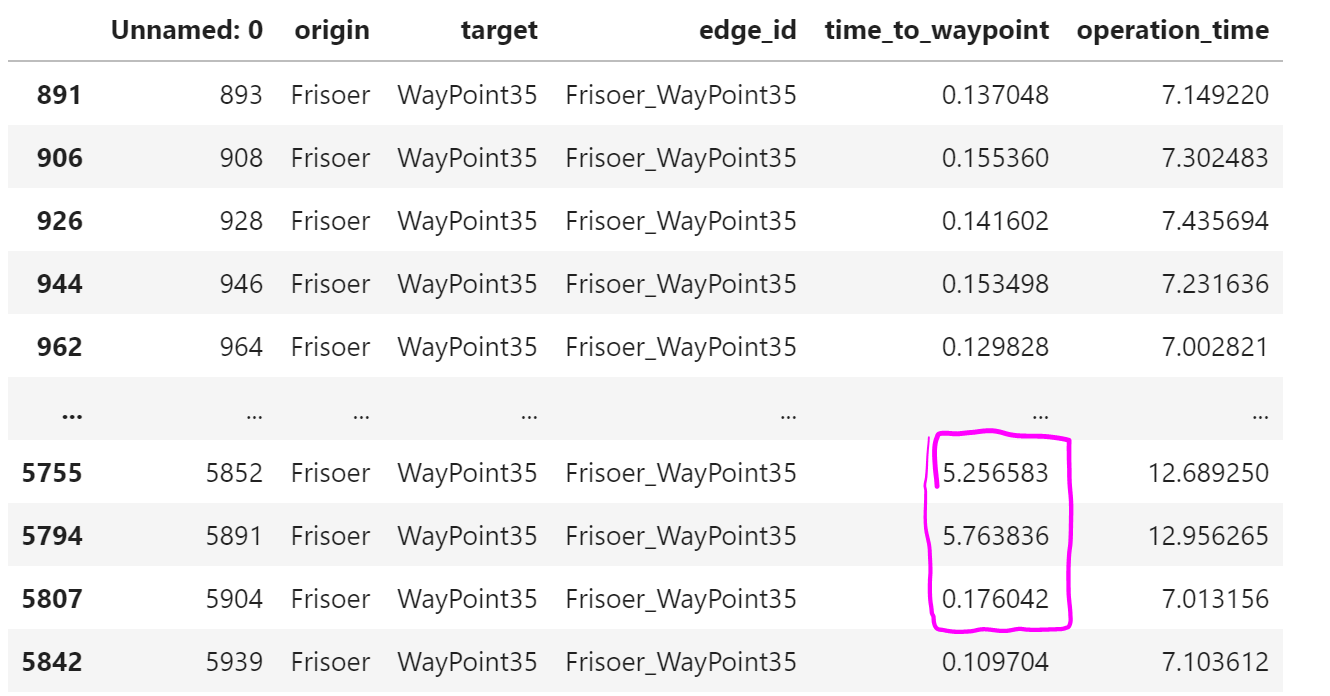
Issue with time-to-waypoint

Date\_finished – date\_started = operation\_time

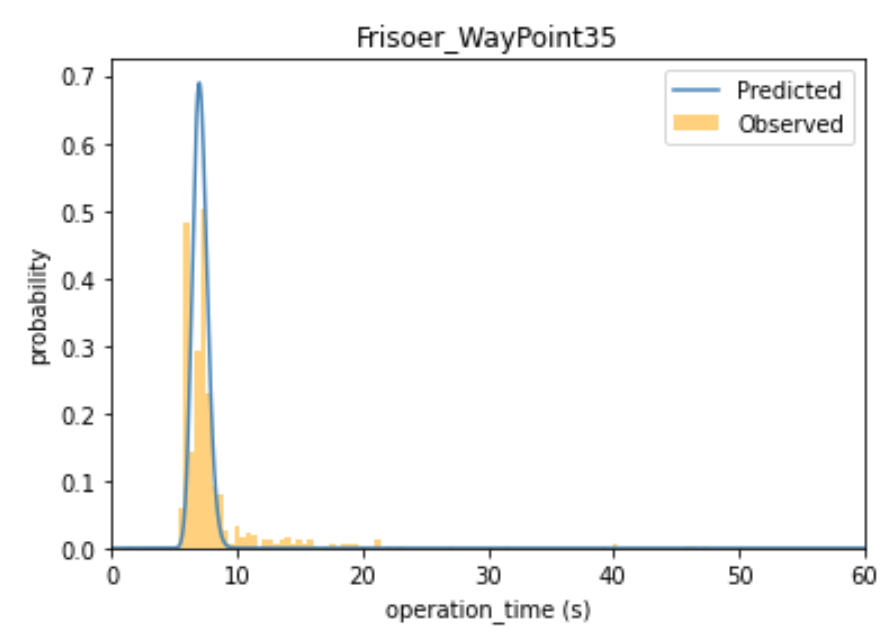
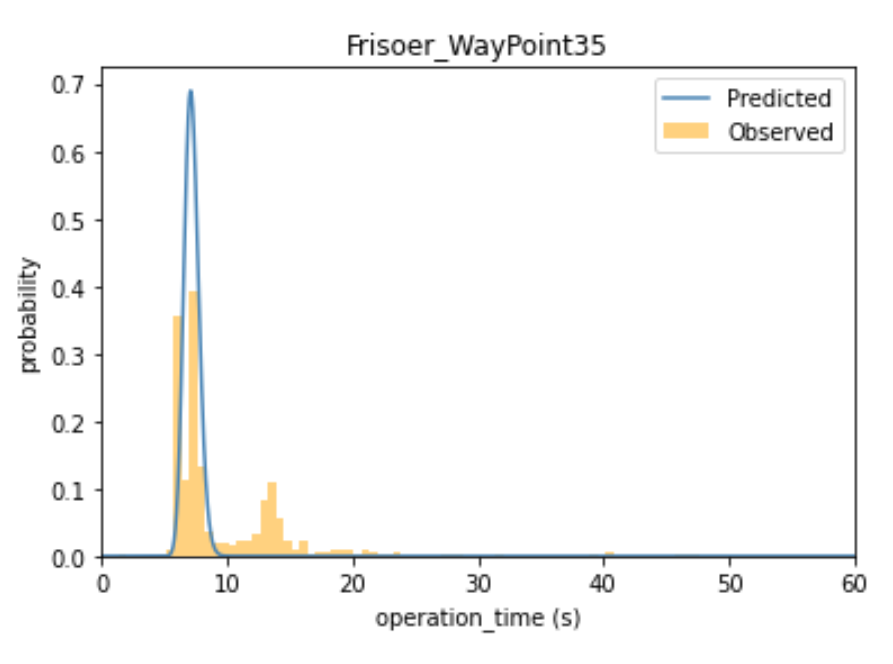
Date\_finished – date\_at\_node = time\_to\_waypoint

**Is time\_to\_waypoint how long it takes for the robot to reorient itself (e.g. turn) once it gets to the target node?**

When fitting operation\_time, we see multi-modality since some instances of time\_to\_waypoint are significantly longer (e.g. for Frisoer\_WayPoint35):

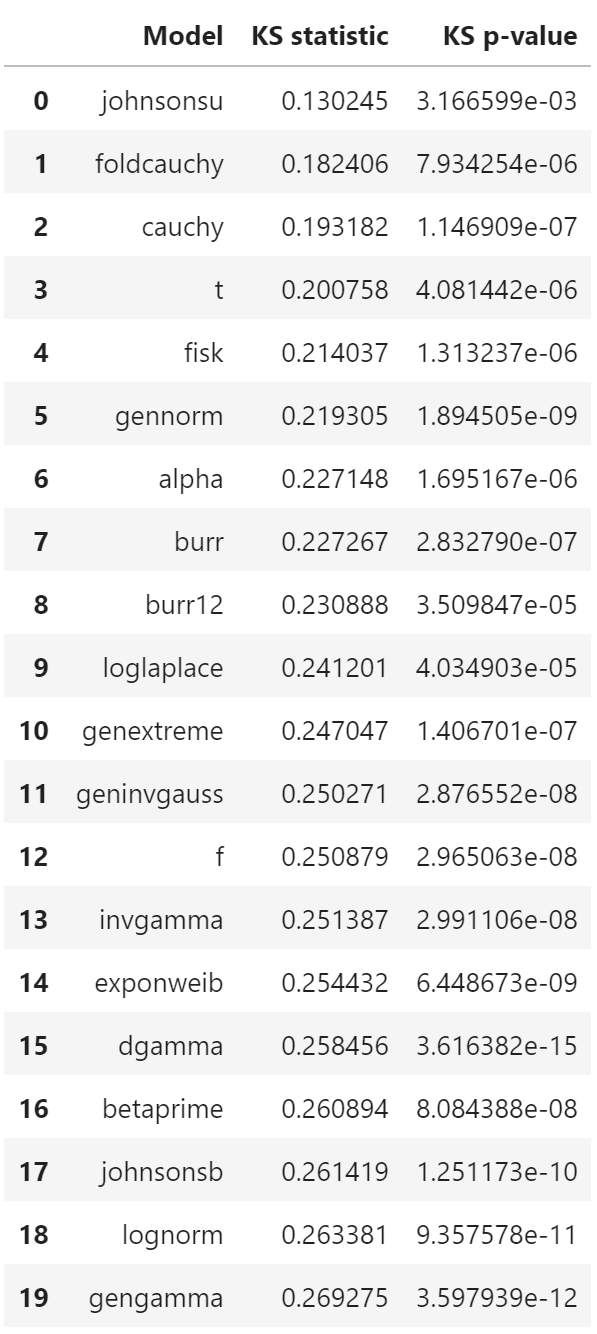
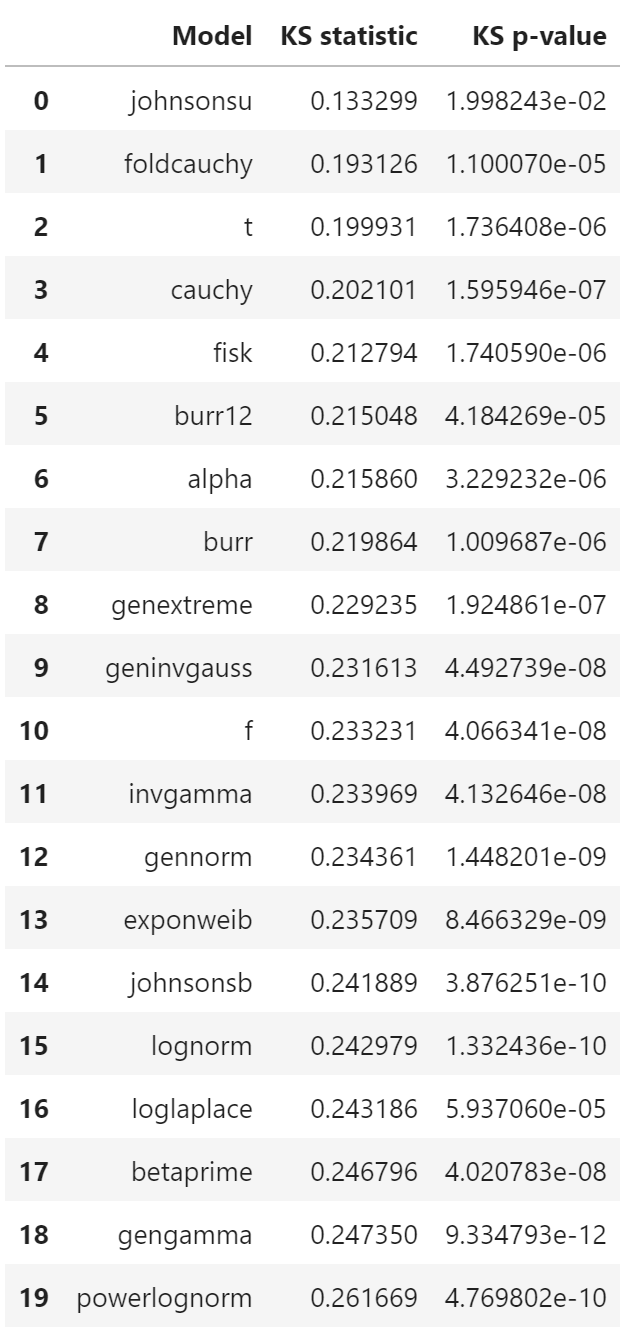


Compare operation\_time (left) to operation\_time – time\_to\_waypoint (right):



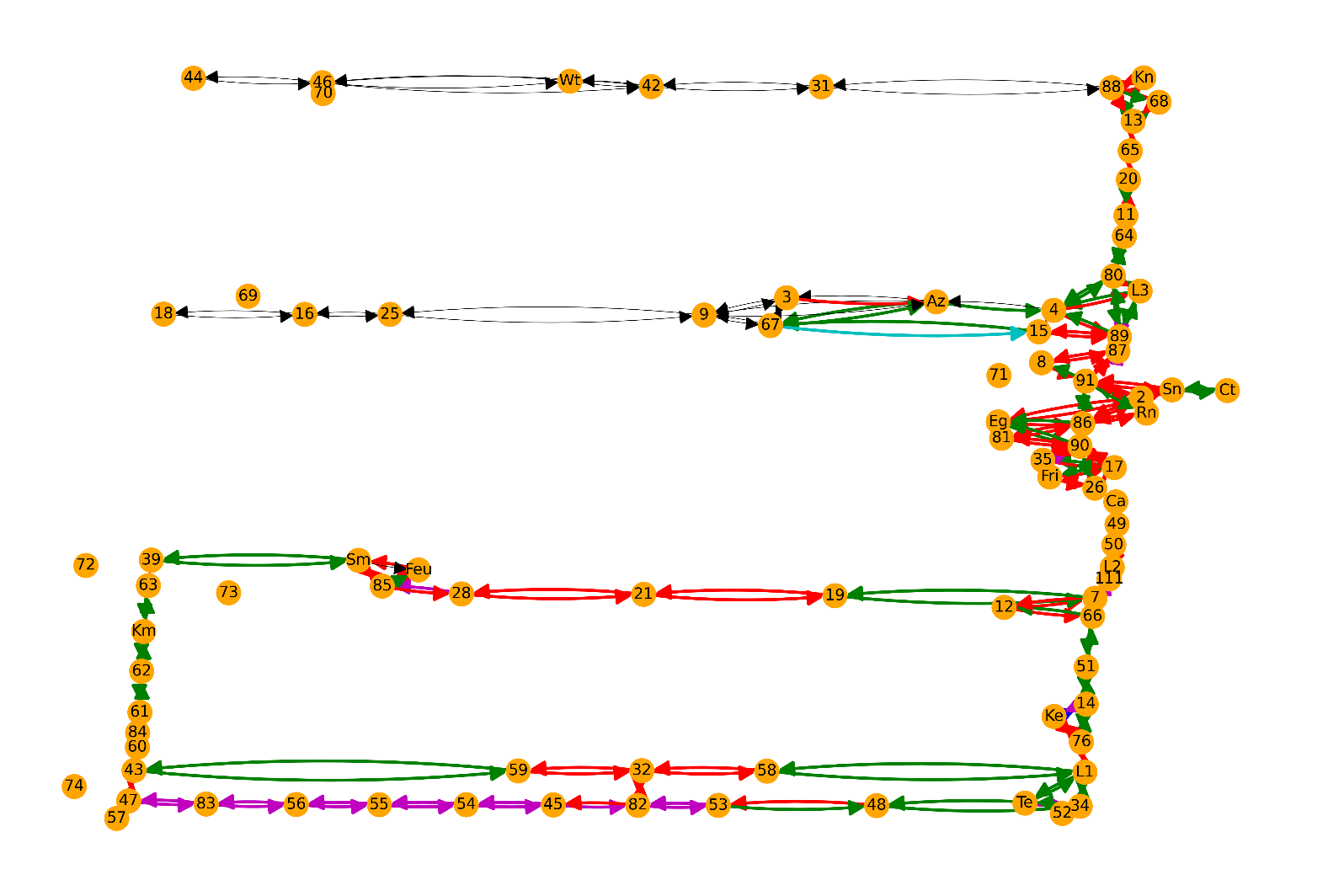
#### GOF

This makes little difference to goodness of fit (left is operation\_time, right is operation\_time – time\_to\_waypoint):

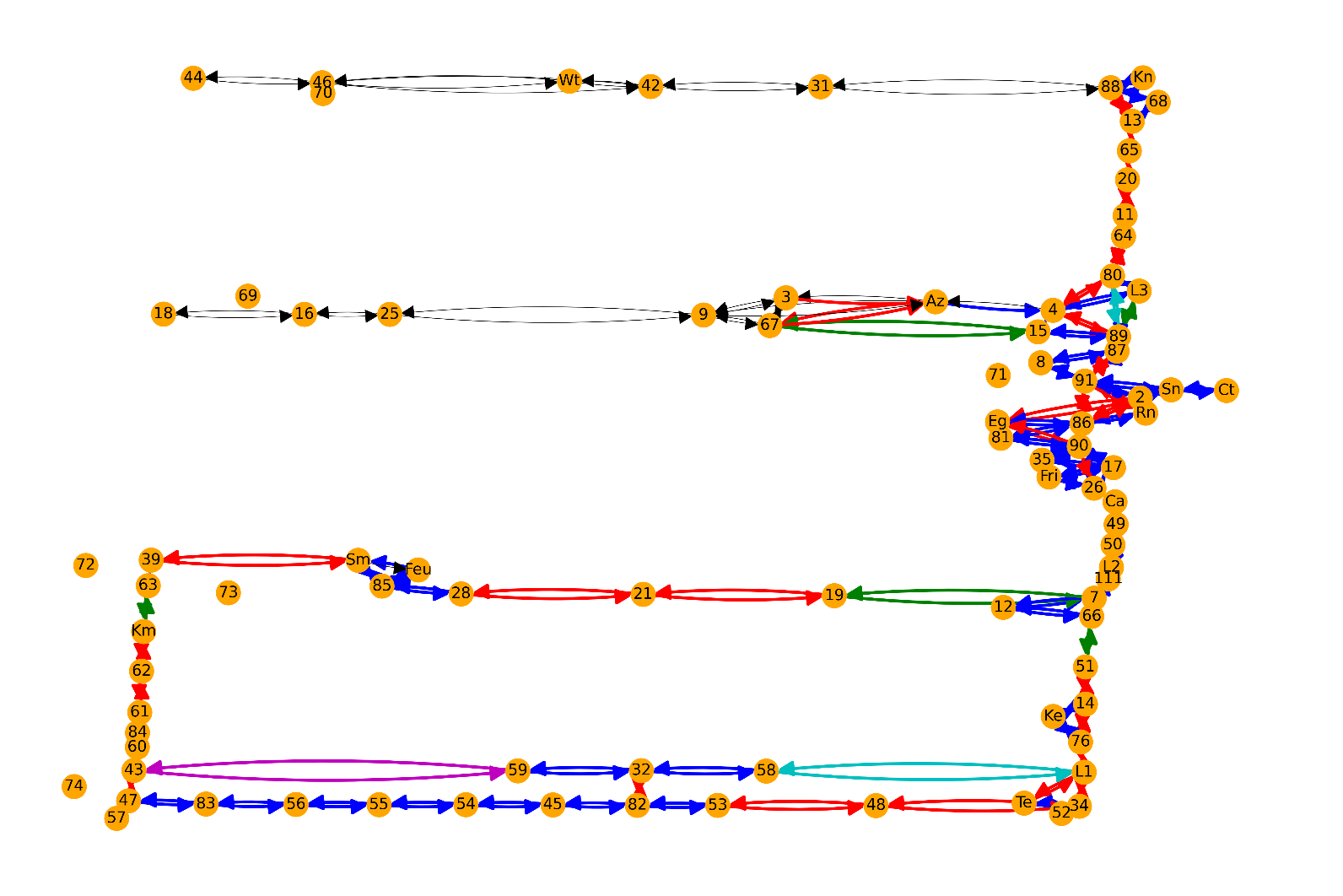


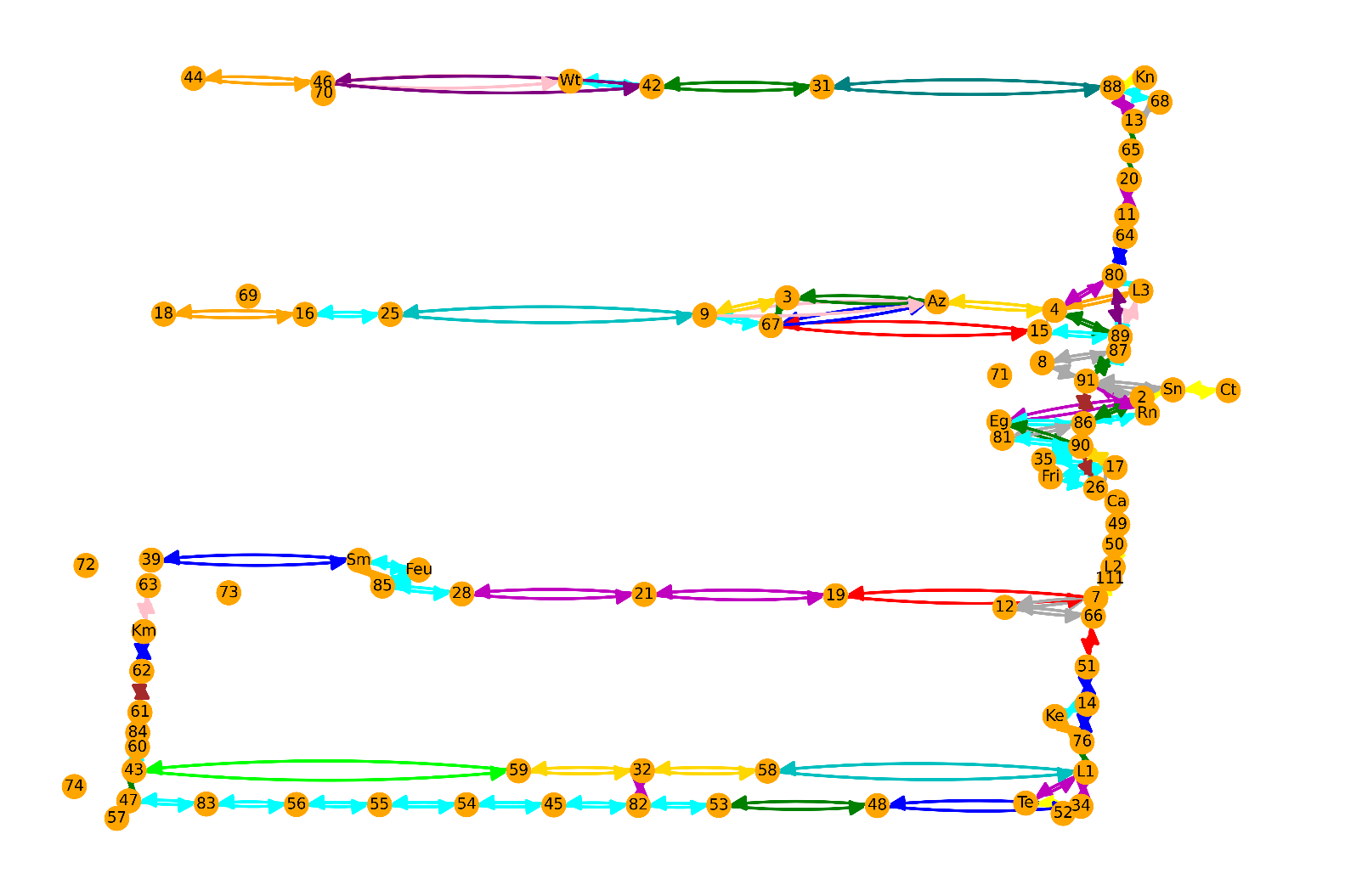
# WED: Clustering

#### By KS

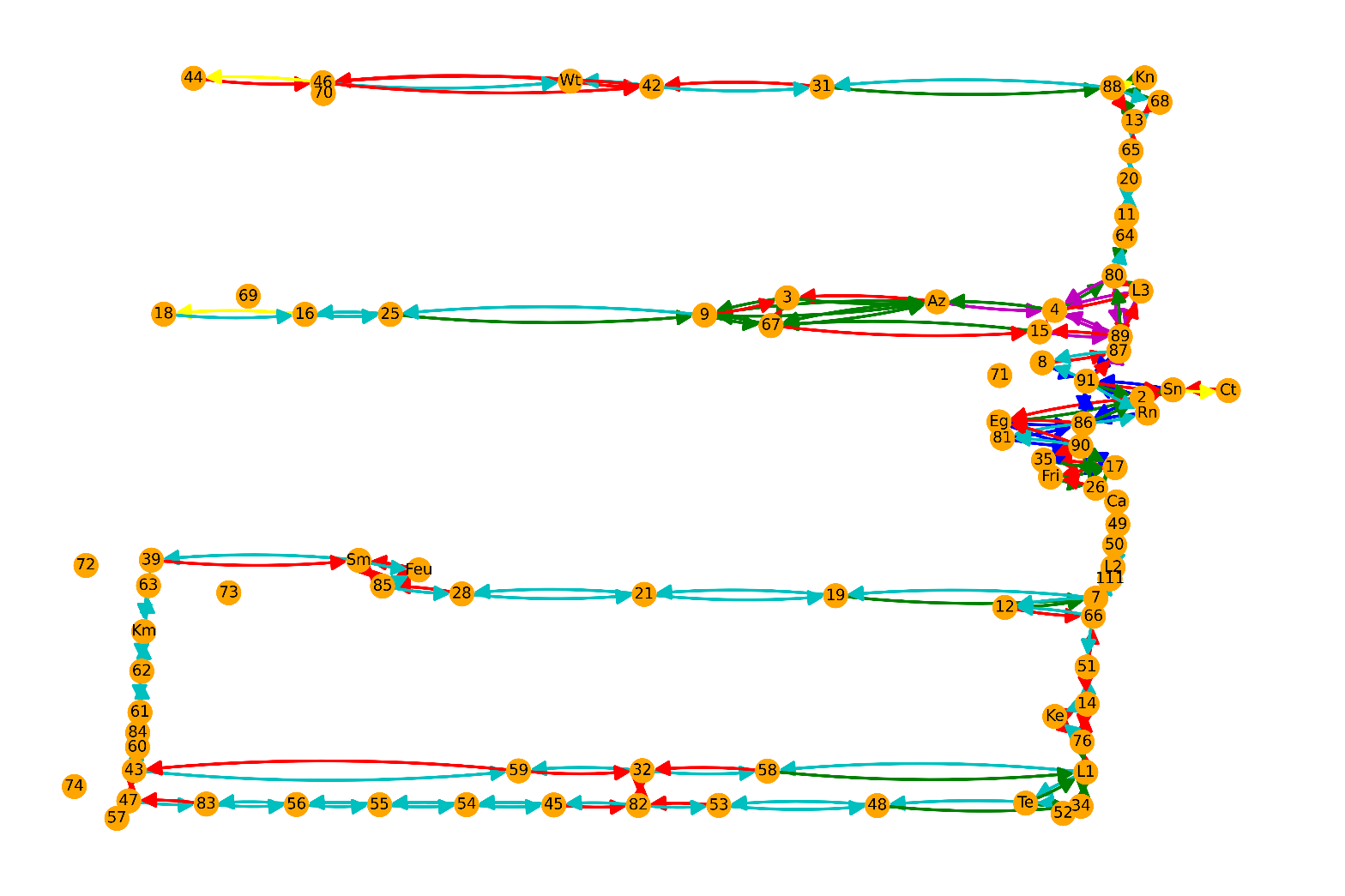


#### By Edge Length (only for edges with KS data / for all edges)

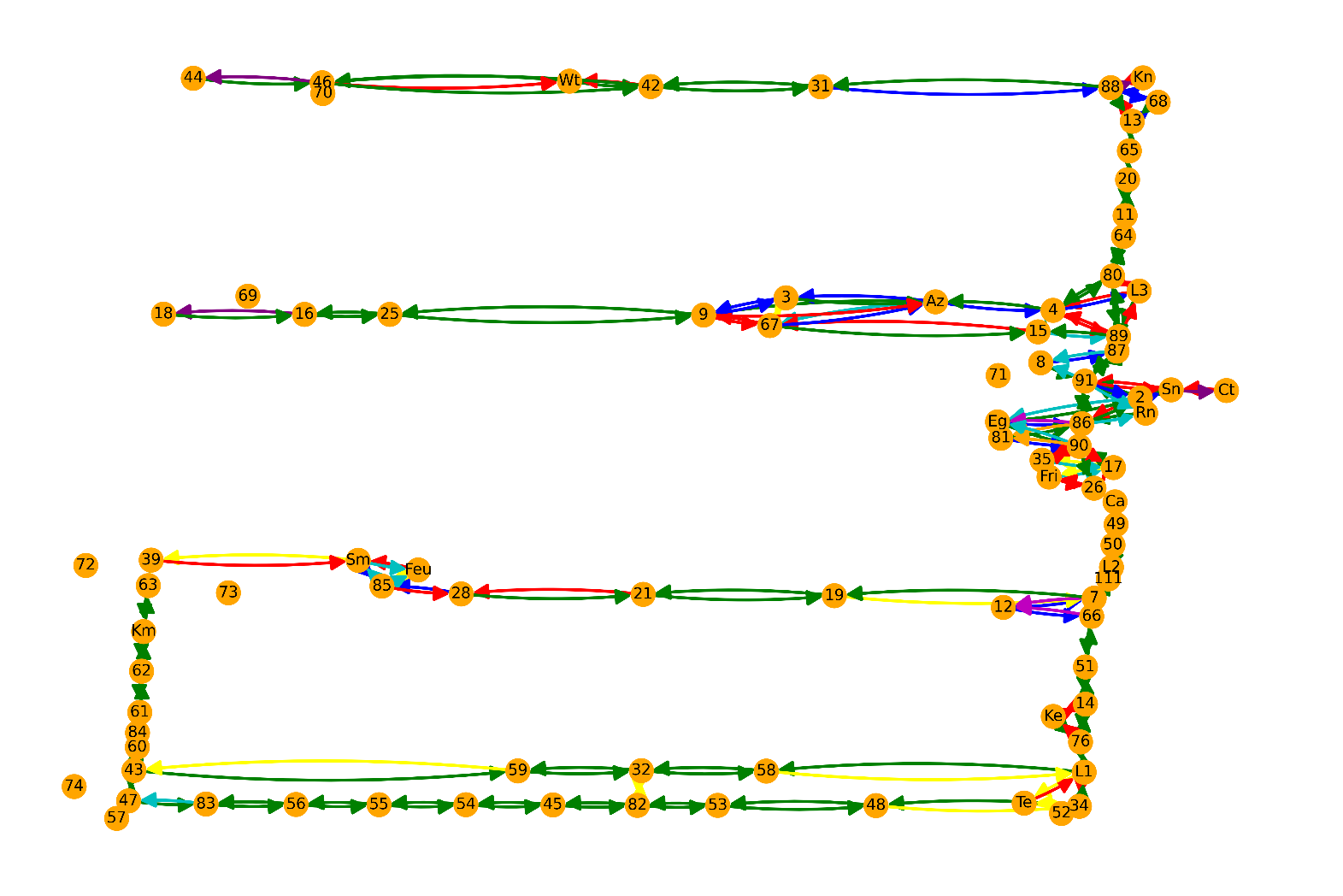




#### By Connections



#### By Angle



# THU: Logistic Regression

2 possibilities for input, output pairings.

**First possibility:**

* **Input:** edge characteristics
* **Output:** Whether an edge is in a specified cluster

**Second possibility:**

* **Input:** difference in edge characteristics
* **Output:** Whether 2 edges are in the same cluster

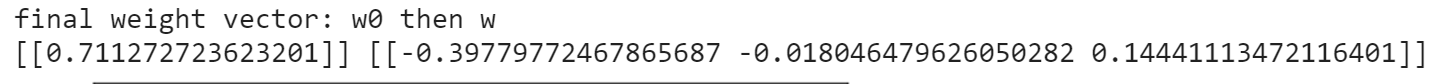
**Why do we choose possibility 2?**

* More datapoints for the same number of observations.
  + For N observations, we have N datapoints if we consider only the edge characteristics.
  + We have N(N-1)/2 datapoints if we consider the DIFFERENCE in edge characteristics.
* Weights vary between clusters
* Weights vary based on which datapoints are used in training’

**Note:** I removed approx. 2-3K datapoints so that the dataset is balanced

#### My implementation of possibility 2

With 1,000 datapoints for training. 13391 datapoints for testing

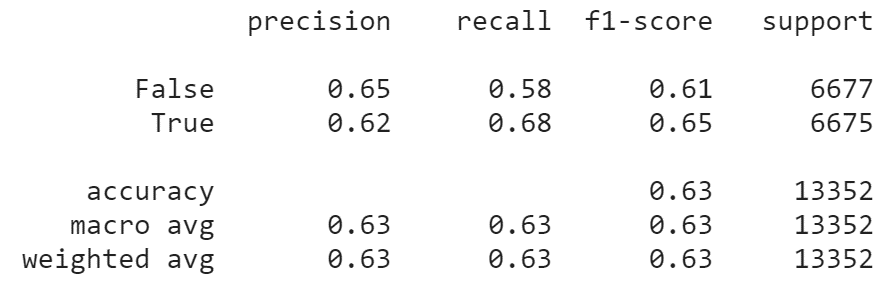


h = w0 + w\*X

y\_hat = sigmoid(h) = 1/(1+exp(-h))

If h is very positive, y\_hat 🡺 1

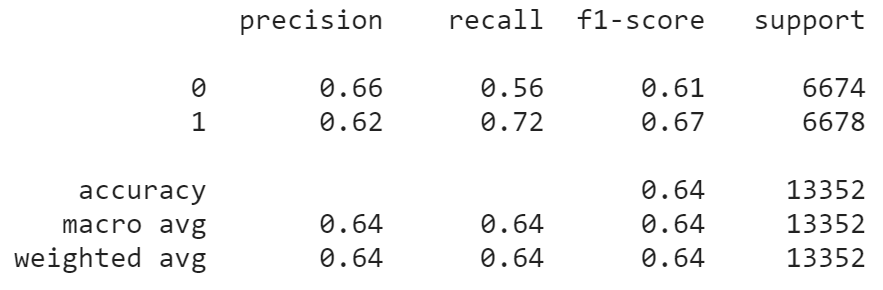
If h is very negative, y\_hat 🡺 0



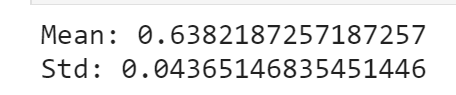
A completely naïve classifier gets 50% accuracy. So this model is not very good

#### Using Cross validation & Sklearn

Slightly better scores – 64% accurate with 1,000 training datapoints. Also much faster to compute.



Cross validation with 100 splits gives accuracy of



### Adding more observed parameters

Input to the logistic regression is the difference in these parameters between 2 edges

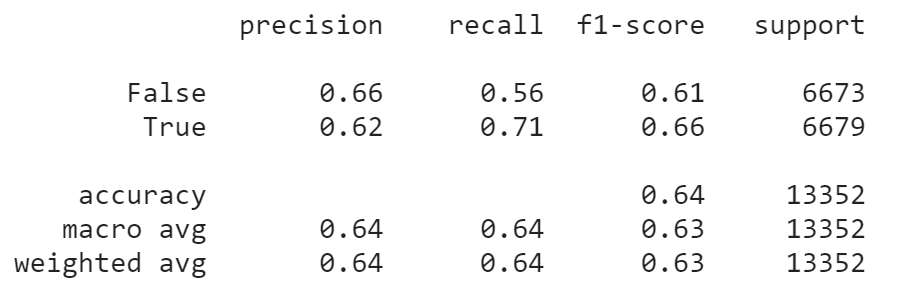
Original:

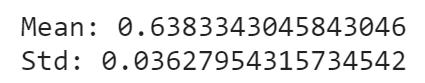
* Edge length
* Target connection
* Max angle of turning

Additional:

* Origin connections
* Total connections
* Sum of angles of turning

**No marked improvement:**



100-fold cross-validation  


# FRI: Neural Networks

Tried alternatives to logistic regression for classification. NNs offer no improvement

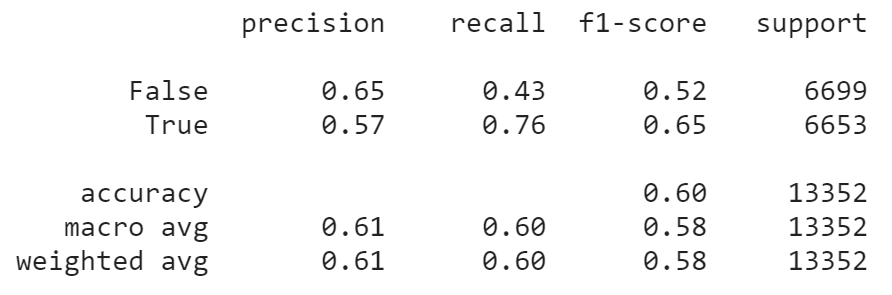
Still need to try:

* SVM
* Random forest
* KNN

#### NN (my implementation) – single layer

10 hidden nodes, 1 hidden layer, 1500 forward/back passes

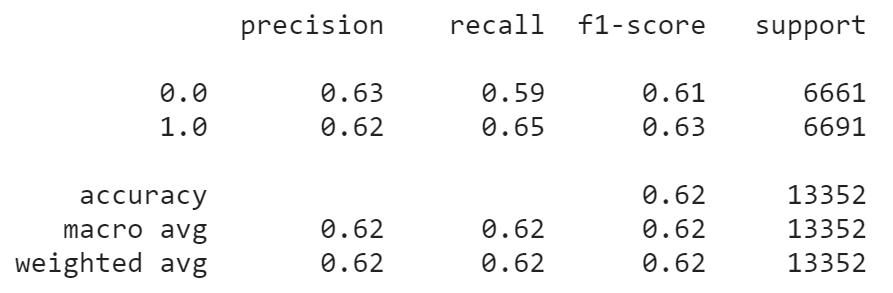
1000 datapoints for training



#### NN (SGD optimise) - one hidden layer

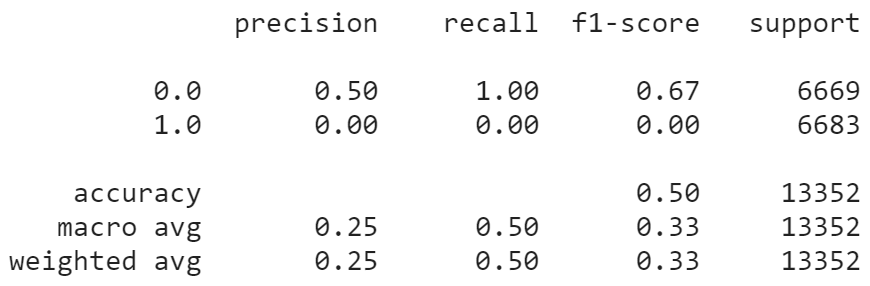
100 hidden nodes, 1 hidden layer, 1000 forward/back passes

1000 datapoints for training



#### NN – more hidden layers

Worse performance, e.g. with 3 layers

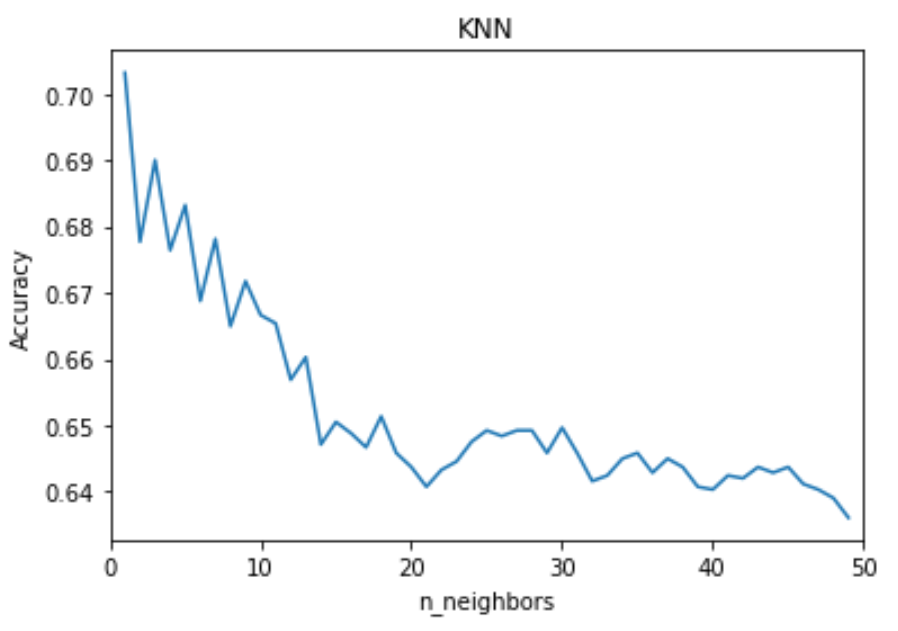
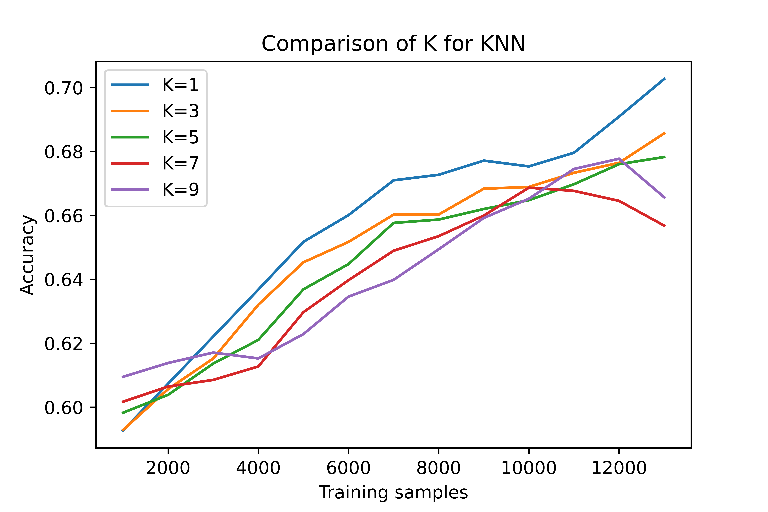


# MON: More Classification methods

#### KNN

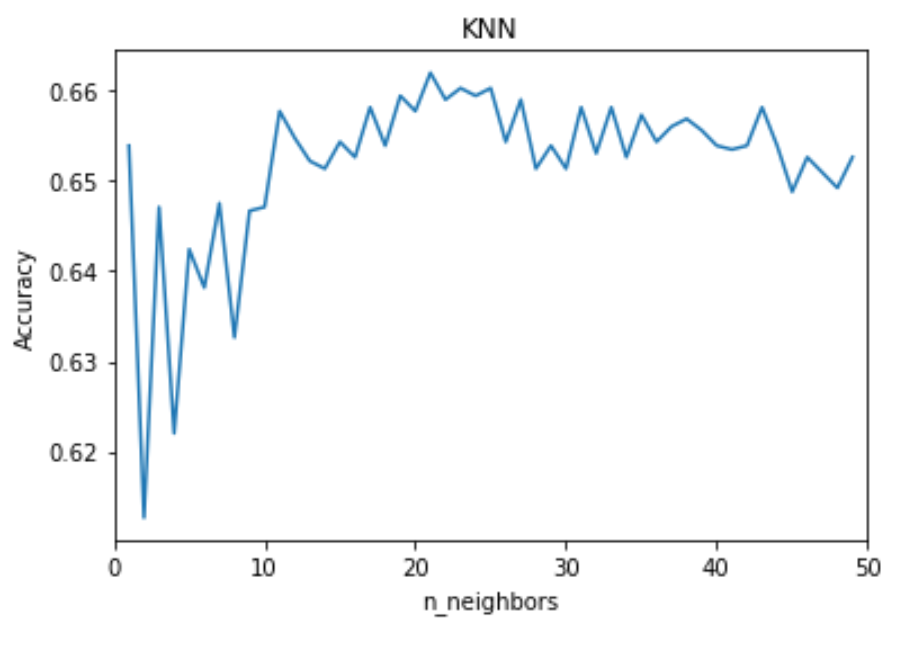
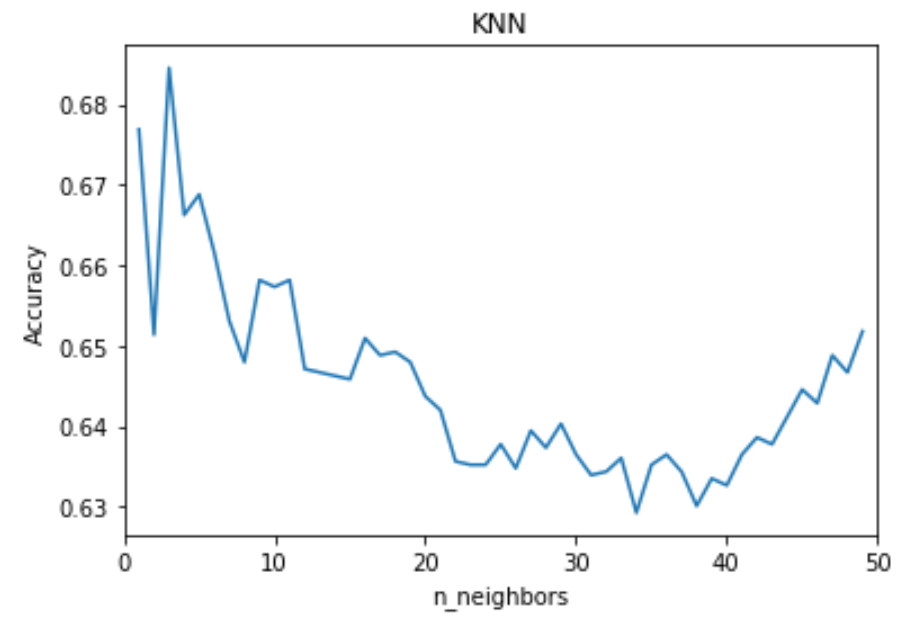
[KNN Classification using Scikit-learn - DataCamp](https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn)

**K = 1 is the best**



**Feature normalisation (left) and reducing no. of features (right) both decrease performance**

Feature importance is evaluated using Random Forest (see later section)

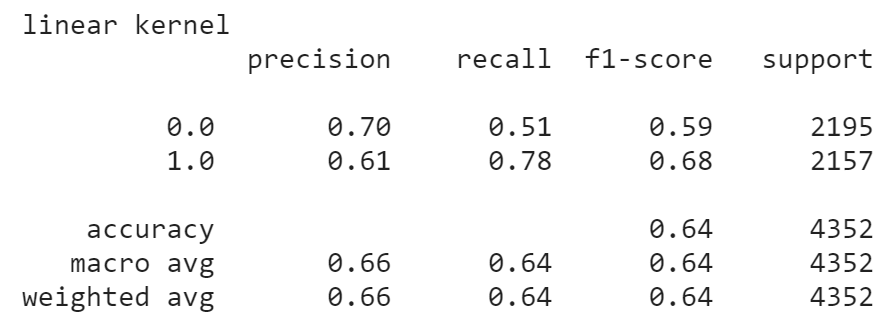
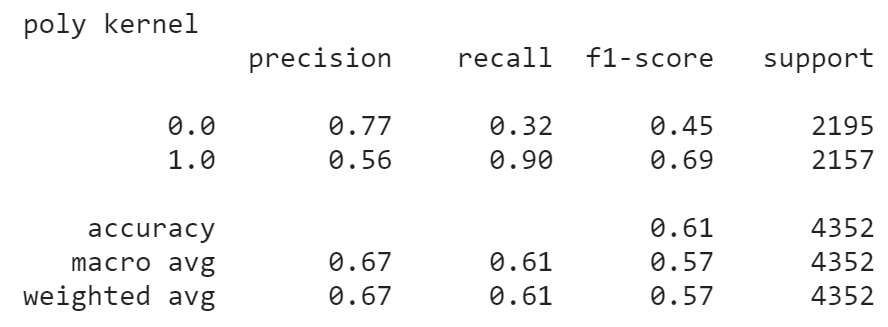


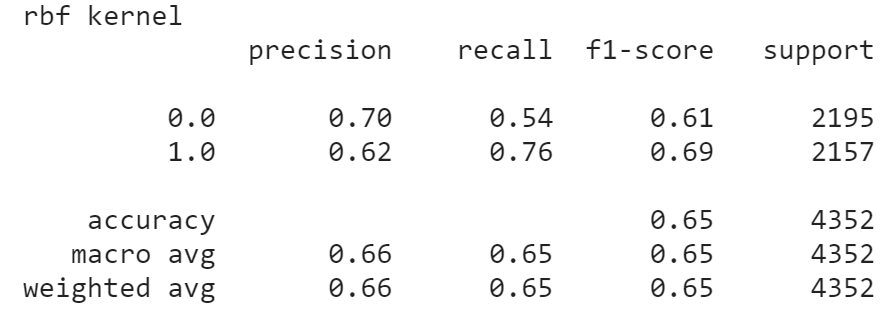
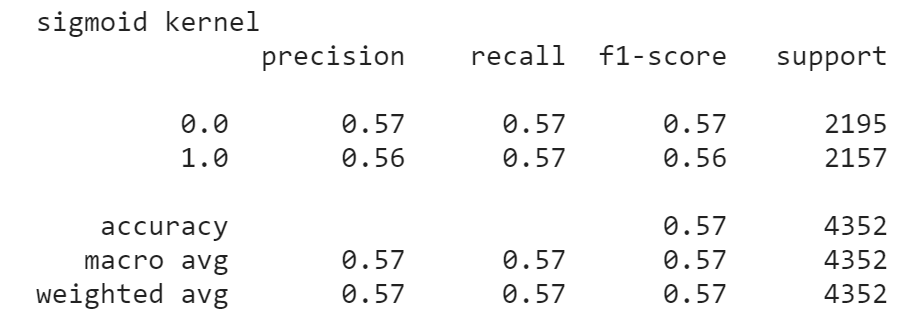
#### SVM

[(Tutorial) Support Vector Machines (SVM) in Scikit-learn - DataCamp](https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python)

[sklearn.svm.SVC — scikit-learn 0.24.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

Radial Basis Function (rbf) method has best performance

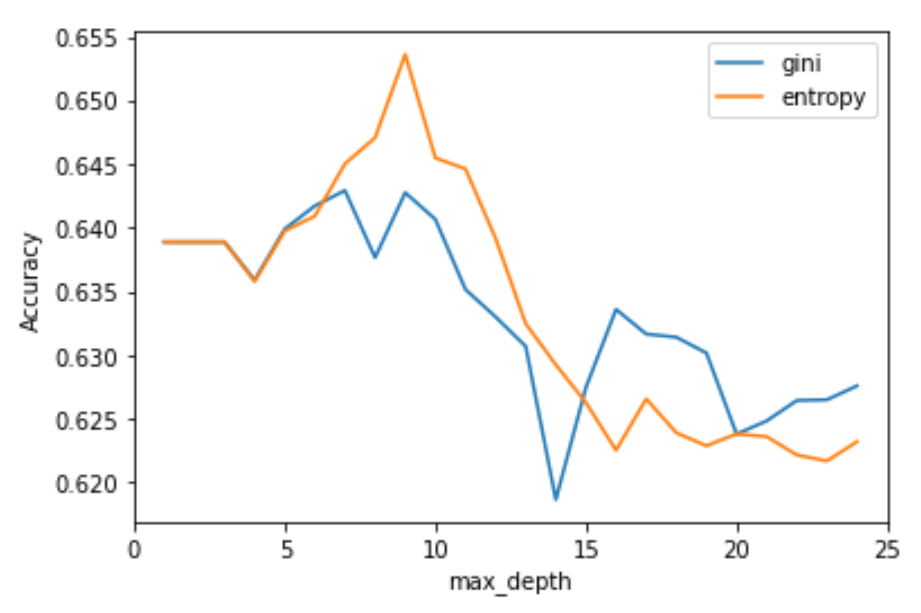
#### Decision Tree

[Python Decision Tree Classification with Scikit-Learn DecisionTreeClassifier - DataCamp](https://www.datacamp.com/community/tutorials/decision-tree-classification-python)

[sklearn.tree.DecisionTreeClassifier — scikit-learn 0.24.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

Gini vs entropy criterion have comparable performance (see Random Forest section)

Max\_depth of **5-10** or **None** do well.

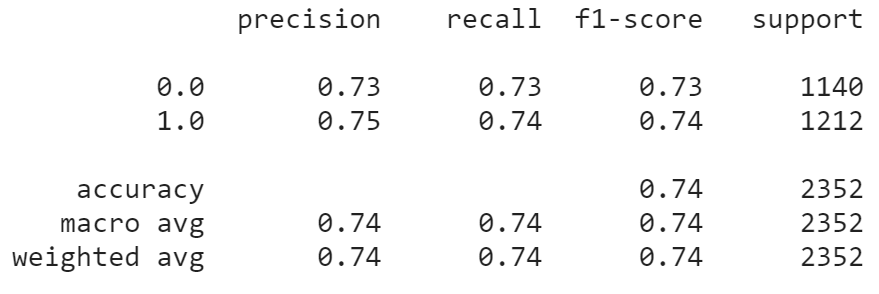


#### Random Forest

[Random Forests Classifiers in Python - DataCamp](https://www.datacamp.com/community/tutorials/random-forests-classifier-python)

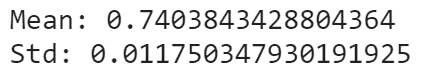
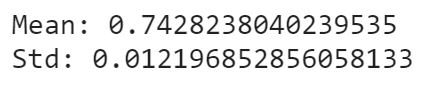
[sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

**RF gives high accuracy and f1 scores. For 12000 samples:**

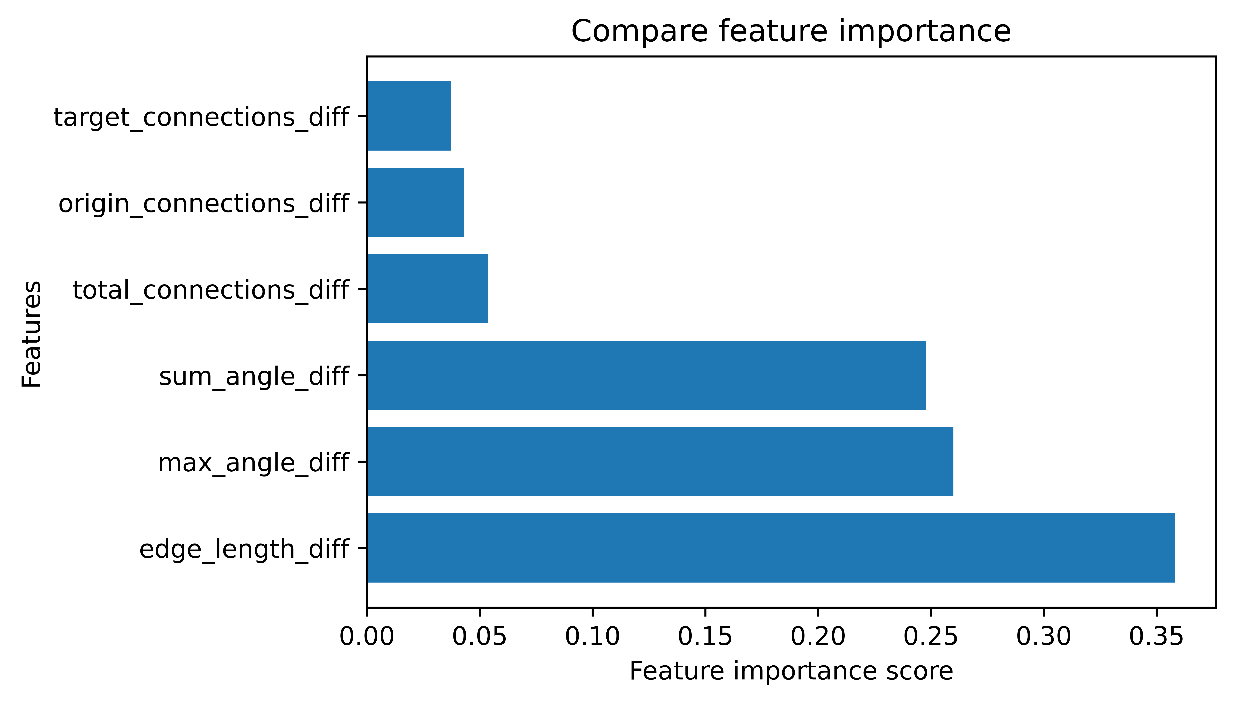


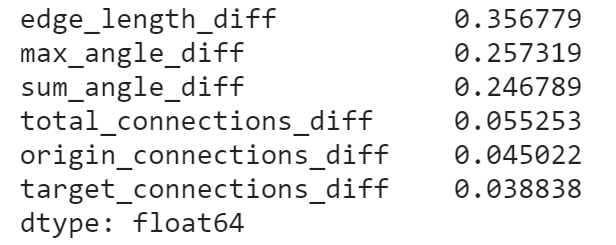
**10-Fold Cross Validation of RF (for 12000 samples) gives accuracy of**

Left (gini criterion): Right (entropy criterion)

**Feature importance: from Random Forest analysis**

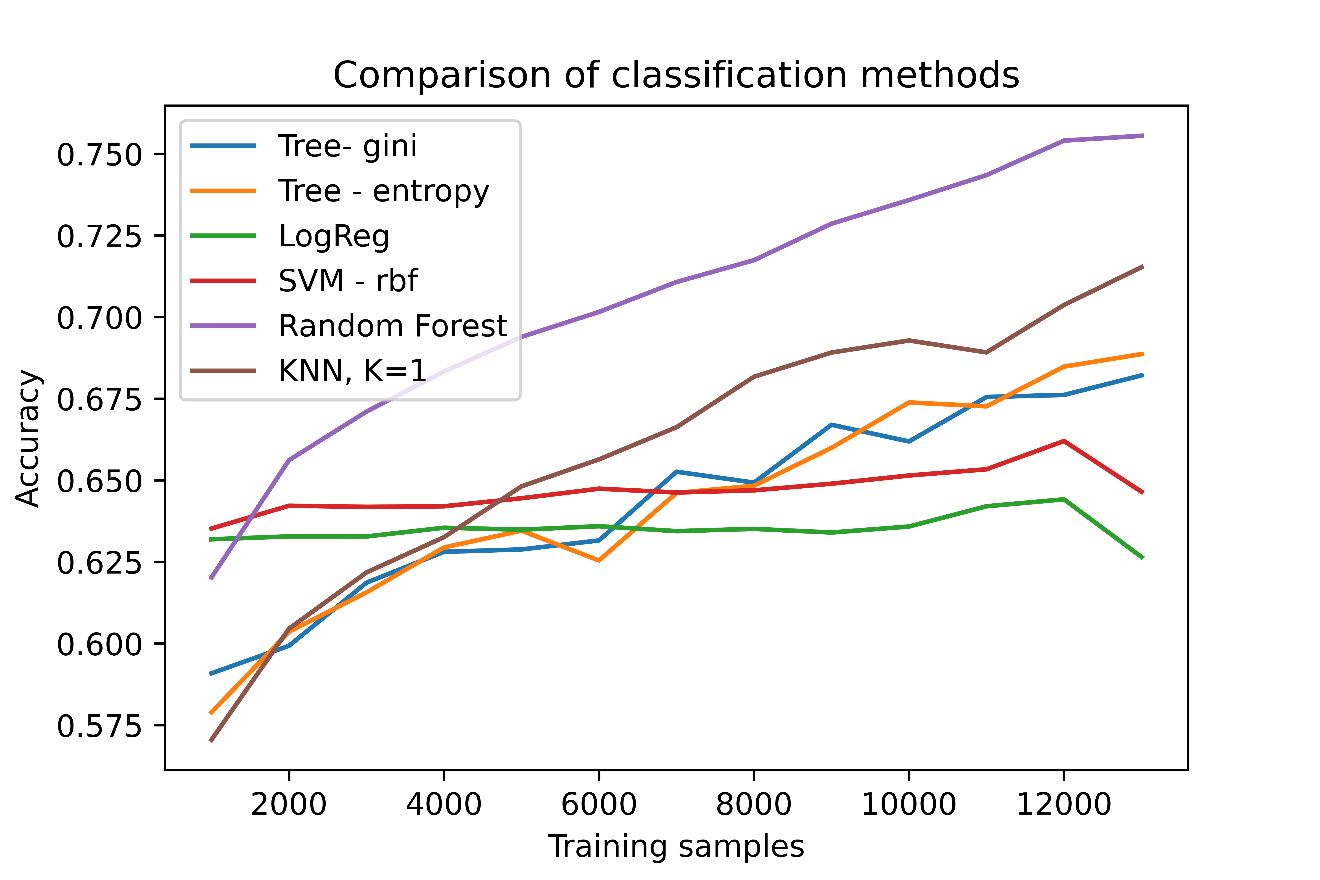




#### Compare Classification Methods

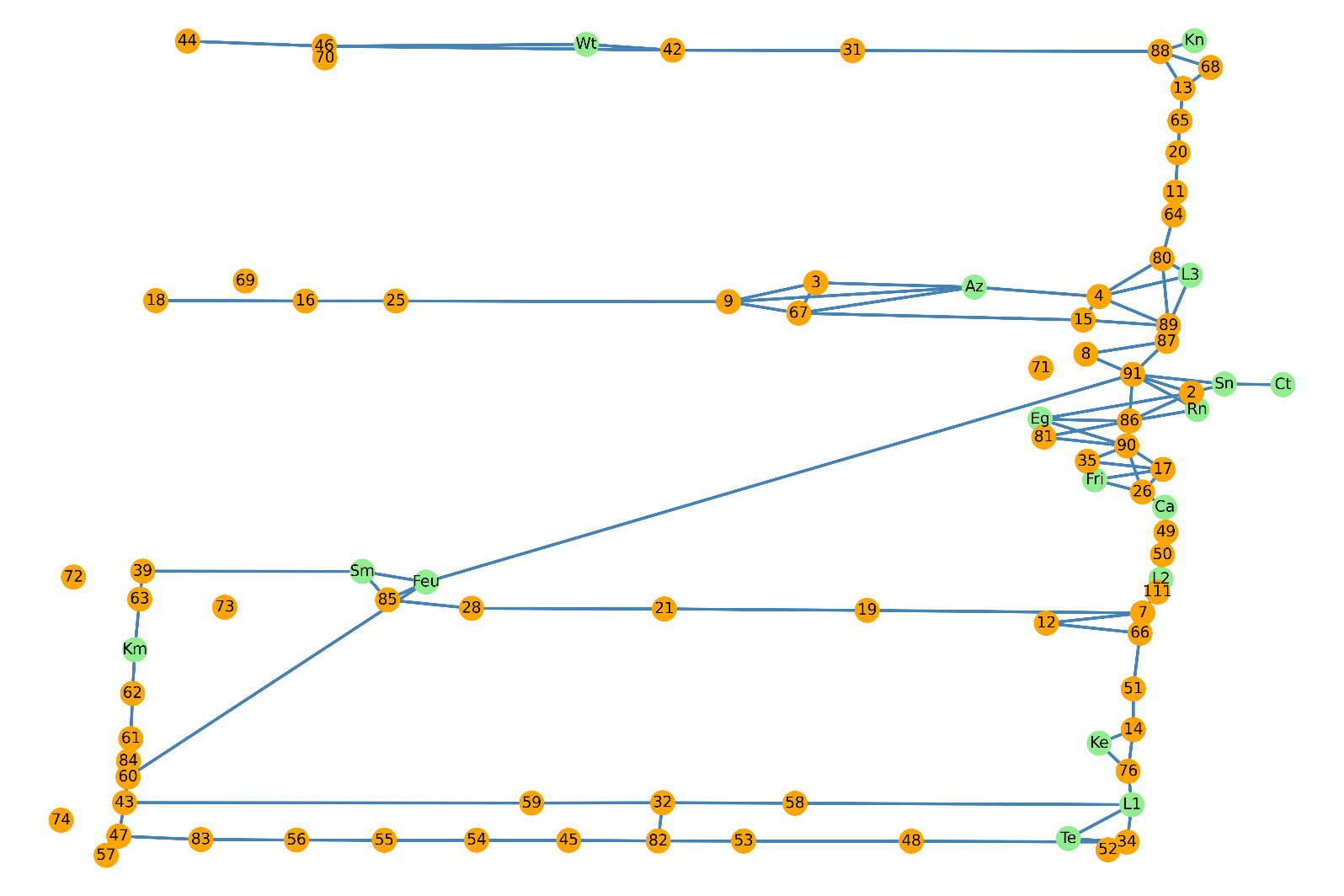
**Accuracies are the averages over 5 iterations:**

* **Best:** Random Forest
* **Improve with training samples:** KNN (K=1), Decision Tree (Gini or Entropy)
* **Performance stays constant (bad):** SVM, Logistic Regression

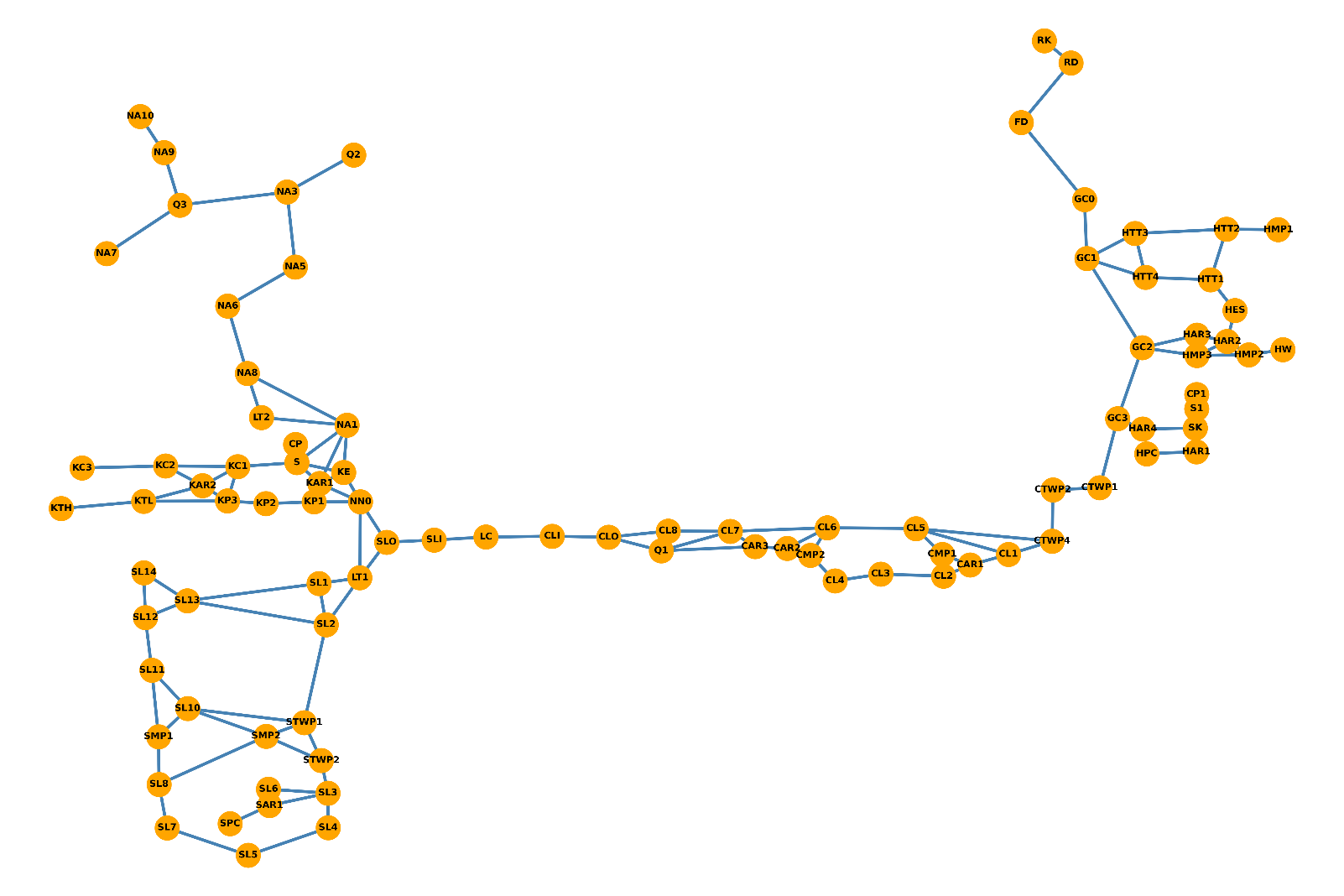


# Maps

**AAF:**



**TSC:**



**LABS:**

