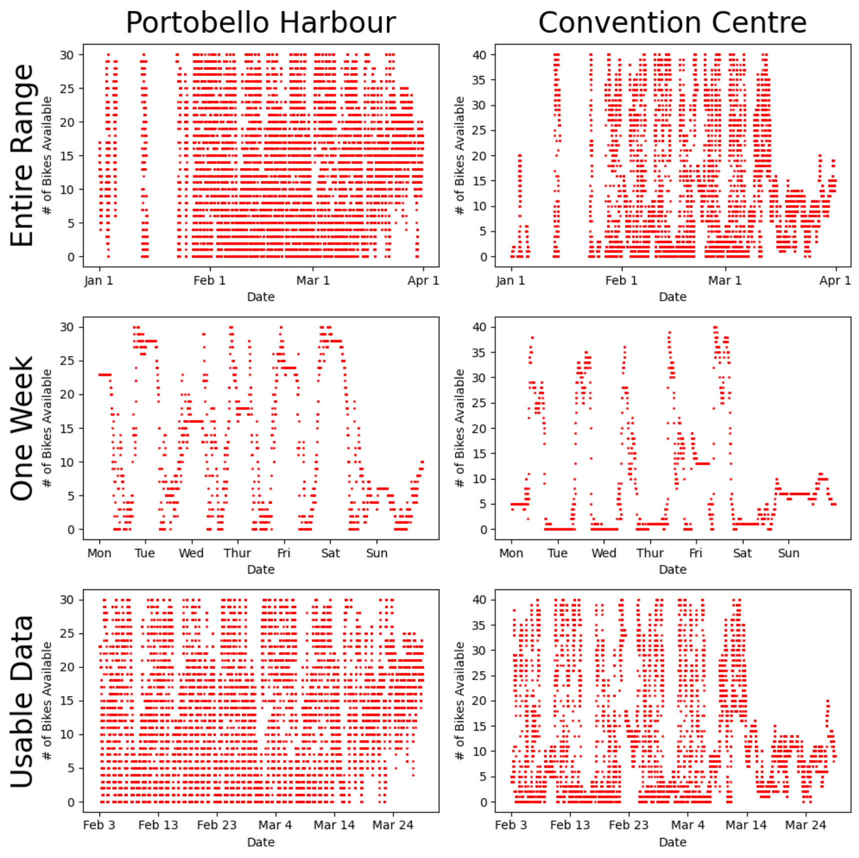
Dataset

For this assignment I have decided to look at the bike stations of Portobello Harbour (station #34) and the Convention Centre (station #65). Portobello Harbour is a primarily residential area whilst the Convention Centre bike station serves a much more commercial and corporate part of the city centre. The Convention Centre bike station is also right next to a Luas stop and only a couple minutes' walk away from Connolly Station. By comparison Portobello Harbour is about 10 minutes away from the nearest Luas stop and 15-20 minutes away from the nearest DART station. Plotting these datasets seems to show that these differences result in different behaviours between the two stations.



One Week graphs cover range from 03 Feb to 09 Feb

Zooming into the data and inspecting the use of each station over the course of a week shows that Portobello is occupied frequently even during the weekend with fairly consistent use throughout the day, whilst the convention centre is used less frequently with spikes during the workday and almost no usage on the weekend.

The dataset covers a timespan from the 1st of January to the 1st of April. However, the data for January is missing large chunks of data, and after the coronavirus lockdowns came into effect around mid-March, there is a sizable and unrepresentative shift in the usage of the bikes. Because of the former issue here, I am going to begin my dataset on Monday, 03 February. The latter issue presents a trade-off between the quality of our data and the quantity of our data. Due to the feature engineering described below our usable dataset becomes even smaller, making the quantity of data available very important. Because of this I decided to end the dataset on Sunday, the 29th of March. Beginning on a Monday and ending on a Sunday should help to ensure that each of the day of the week is represented equally within our data.

Linear Regression – Portobello Harbour

Linear regression is often the first go to type of model that works well on many problems. It is fast and easy to train with lots of room to adapt to different problems through feature engineering. Regularisation techniques can also be applied to avoid overfitting.

Feature Engineering

Timestamps

The timestamp format of the given dataset is not very convenient to work with and isn't in a format that can be easily understood by a machine learning model (yyyy-MM-dd hh:mm:ss). To combat this, I decided to convert the timestamps into Unix time, since this format makes it easy to process the order of a sequence of timestamps and the scale of gaps between them. I then normalised the feature by shifting it so that the dataset starts at 0, and the number represents how many days (and how much of a day) we are from that origin.

Trend vs Seasonality & Feedforward vs Feedback

From looking at the graphs on the previous page, it seems like it is important to know both what the current state of the bike station is, as well as how it usually behaves at this time. There appear to be three major factors that would likely help us to make accurate predictions. These are as follows:

1. The current state of bike station occupancy
2. The usual state of bike station occupancy at this time of day
3. The usual state of bike station occupancy on this day of the week

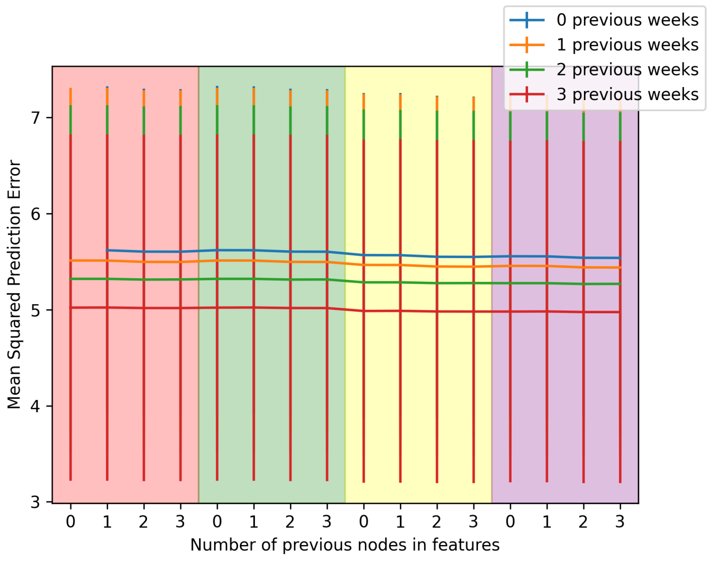
However, it is important for us to determine how many nodes we should go back and include for each of these factors to accurately capture this behaviour without overfitting to the given data. It's also important that since we need multiple previous weeks of data, the more previous weeks we require, the more of our dataset is going to be rendered unusable due to not having the necessary previous data itself.

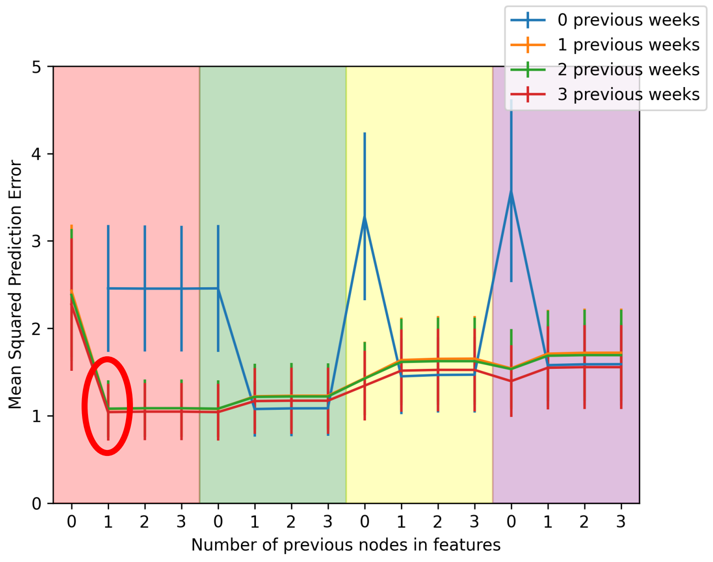
I decided to create a series of models to predict the number of bikes available at Portobello Harbour 10 minutes ahead, using hyperparameter C = 1 and L2 regularisation as preliminary values. I then created multiple instances of the features data in the following format for various values of x, y and z:

```
[timestamp,
value at x previous nodes,
value at this time node for y previous days,
value at this day+time node for z previous weeks]
```

I first trained some **feedforward ridge regression models**. To do this I shifted each of the output features the desired amount of nodes back so that each set of features points to the number of bikes that many nodes in the future. I then trained each of the models with a different instance of the feature set and evaluated their performance.

I also trained some **feedback ridge regression models**. This involved repeatedly making new predictions and reintegrating those predictions into the data until we reached the target time. I also trained these models with all of the various feature sets and evaluated them, leading to the following results:





Left: Feedforward Ridge Regression

Right: Feedback Ridge Regression

Each Zone represents how many previous days, y (0, 1, 2 or 3) are included in the feature set

0 nodes, 0 days and 0 weeks is not included since it's MSPE is so large it makes these graphs unreadable

I chose MSPE as my scoring metric since I want to penalise predictions that are further away from the truth much more than predictions that are closer. As we can see from these results the feedback model performs significantly better with every feature set than the feedforward model, and as such I will be using **feedback models** going forwards. I will also be using the feature set with **1 previous node (x), 0 previous days (y) and 3 previous weeks (z)** since it performed better than most other sets and just as well as the more complicated 2 and 3 previous node ones too.

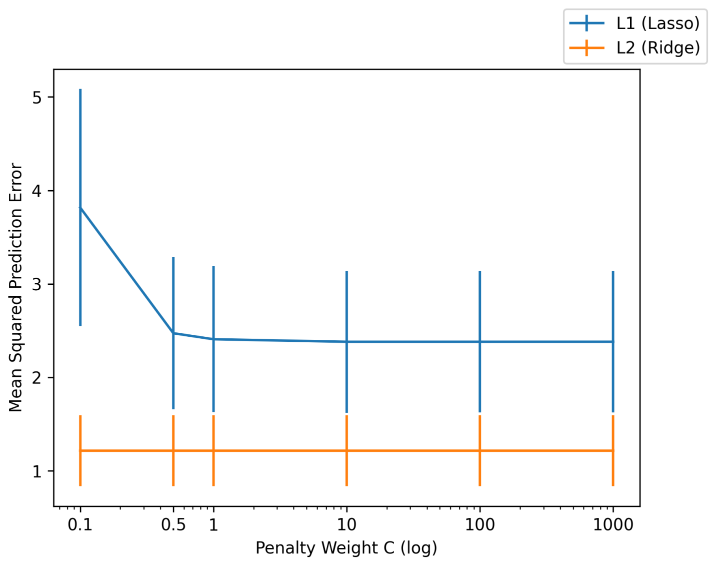
Cross Validation

Penalty Weighting (C) & Regularisation Type

When training a linear model you have to choose between L1 and L2 regularisation. In short, L1 regularisation tries to encourage the model to use as few features as possible, whilst L2 regularisation tries to weight each of the features as lightly as possible. We also have to choose a value for the hyperparameter C, which specifies how significant our penalty will be. The higher the value of C, the less strong our regularisation will be.

I redid the feature engineering process in the previous section with L1 regularisation and found that feedback models and the same feature set remained the most effective predictors.

I trained a series of L1 (Lasso regression) models and L2 (Ridge regression) models using various values for C and the specifications determined above to predict the number of bikes available at Portobello Harbour 10 minutes ahead. I evaluated each of their predictions and got the following results:

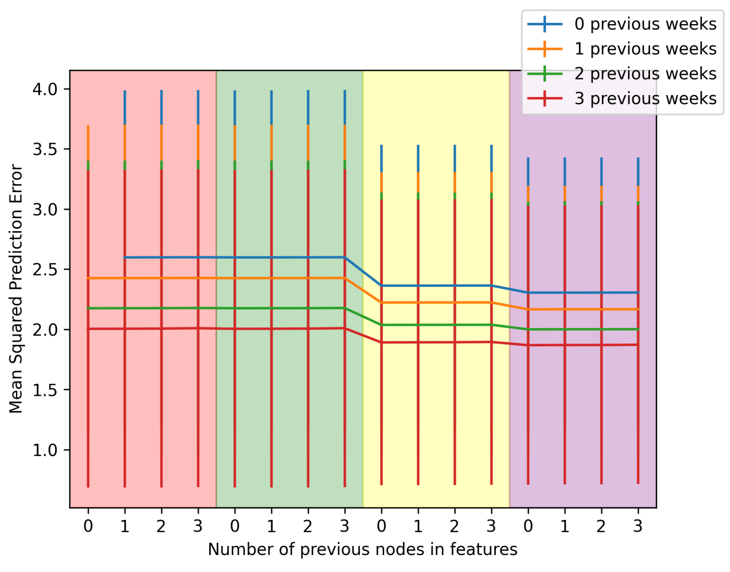


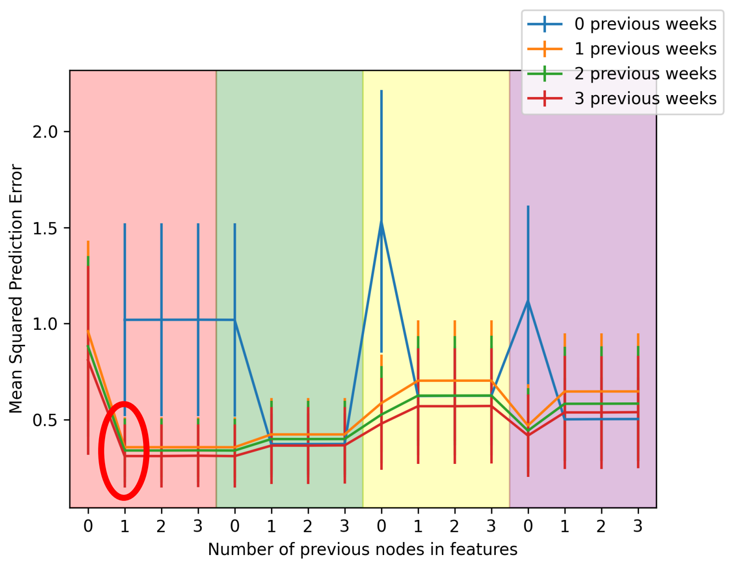
MSPE vs C with L1 and L2

As we can see from these results the L2 regularisation clearly performs better than the L1 regularisation, however the value of C doesn't particularly affect the results. Due to this, I have decided to train the model using **ridge regression** with **C=1**.

Linear Regression – Convention Centre

I repeated the above steps in order to train a linear regression model on the convention centre data.





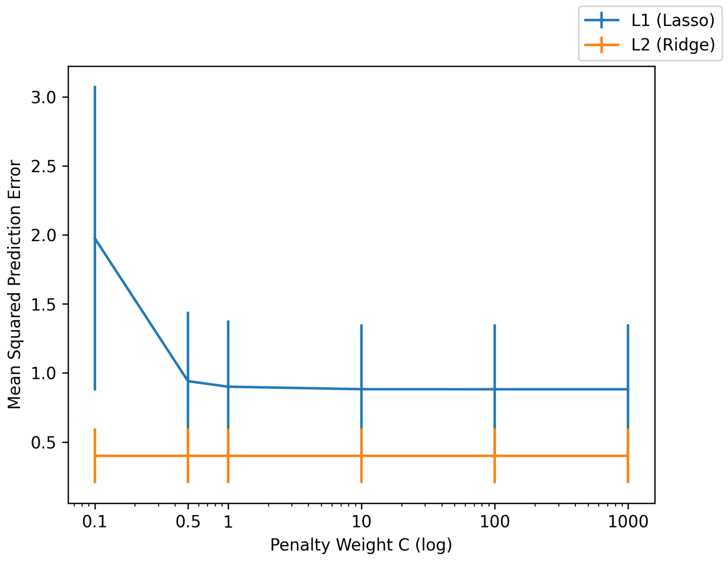
Left: Feedforward Ridge Regression

Right: Feedback Ridge Regression

Each Zone represents how many previous days, y (0, 1, 2 or 3) are included in the feature set

0 nodes, 0 days and 0 weeks is not included since it's MSPE is so large it makes these graphs unreadable

Again, the **feedback model** shows a noticeable improvement over the feedforward model. The best feature set was **1 previous node, 0 previous days & 3 previous weeks** (circled in graph). I tested this with Lasso models (L1 regularisation) and the same choices still produced the best results.



MSPE vs C with L1 and L2

Once again **L2 regularisation** was evidently more effective at all values of C, whilst C didn't really impact our models performance, leading me to stick with **C=1**.

K-Nearest Neighbours – Portobello Harbour

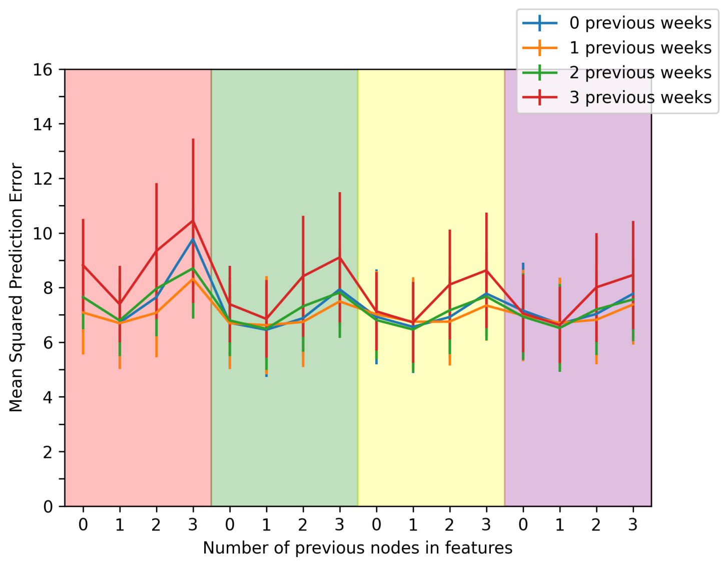
K-Nearest Neighbours usually isn't a type of model that extrapolates well beyond the known dataset, which wouldn't make it an obvious choice of model for predicting into the future. However, I believe that since we have data from every hour of every day of every week without any large gaps in-between datapoints, we can construct a fairly robust K-Nearest Neighbours model. I also had initial concerns about the performance of a K-Nearest Neighbours model on a dataset of this size since the model requires searching the entire dataset K times for every prediction. K-Nearest Neighbours is not a type of model that performs well with larger datasets, however after doing some initial testing on my machine, it ran well enough for me to be comfortable going through with it.

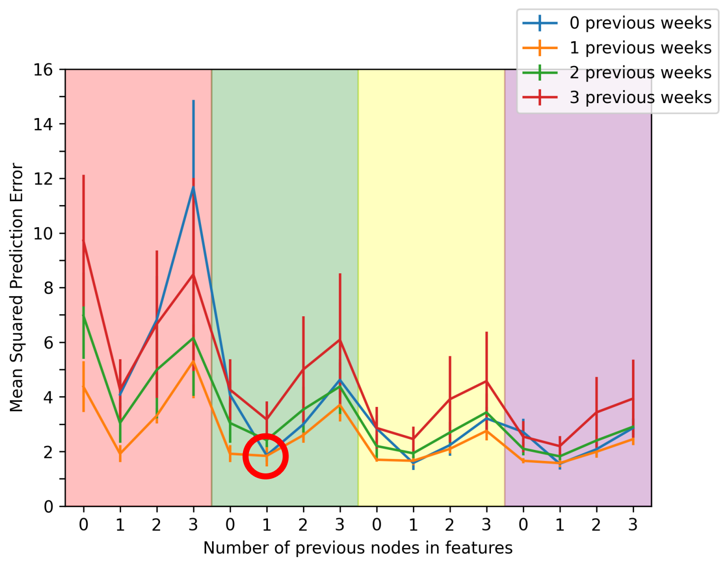
Feature Engineering

I performed the same changes to the feature set for this model as I did for the Linear Regression previously. I made the exact same changes to the timestamp feature and performed the same testing to determine which extra features to include.

Trend vs Seasonality & Feedforward vs Feedback

I decided to create a series of k-NN models to predict the number of bikes available at Portobello Harbour 10 minutes ahead, using hyperparameter k = 20 and uniform weighting as preliminary values. I then created the same sets of features data as before, trained, tested and evaluated the models with them and got the following results:





Left: Feedforward k-NN Regression

Right: Feedback k-NN Regression

Each Zone represents how many previous days, y (0, 1, 2 or 3) are included in the feature set

0 nodes, 0 days and 0 weeks is not included since it's MSPE is so large it makes these graphs unreadable

From this we can see that interestingly the feedback model is not universally better than the feedforward model (unlike the Linear Regression models earlier) but it does improve our best versions of the model so I will be using **feedback** going forwards. Within the feedback model a good option appears to be **1 previous node, 1 previous day & 1 previous week**.

Weekdays and Weekends

From looking at the weekly graph breakdowns in the dataset section at the beginning, it is clear that there is very different behaviour on weekends vs weekdays. However, each of the days within the weekday/ weekend have little and less easily predictable variation between them. I wanted to see if our current model could be improved by adding features to specify which day of the week a given piece of data refers to. There are three possible ways of modelling the difference between days:

1. Do not distinguish between days at all
2. Specify whether or not a specific day is a weekday or weekend
3. Specify which day of the week each day is

Strategy 1 is what we already looked at above, so I decided to change our k-NN models from before to use strategy 2. I modified the features of our data to be like so:

```
[timestamp,
value at x previous nodes, is weekday for each node
value at this time node for previous day, is weekday for each node,
value at this day+time node for z previous weeks, is weekday for each node]
```

I recreated the above plots with this new strategy, and surprisingly it made essentially zero difference to any of the nodes in any of the graphs. I suspect that the previous nodes, days and weeks features are already capturing the nature of the weekday/ weekend difference too well for this feature to matter.

Following this I modified the features again to model strategy 3 instead, like so:

```
[timestamp,
value at x previous nodes, is weekday for each node
value at this time node for previous day, is weekday for each node
value at this day+time node for z previous weeks, is weekday for each node]
```

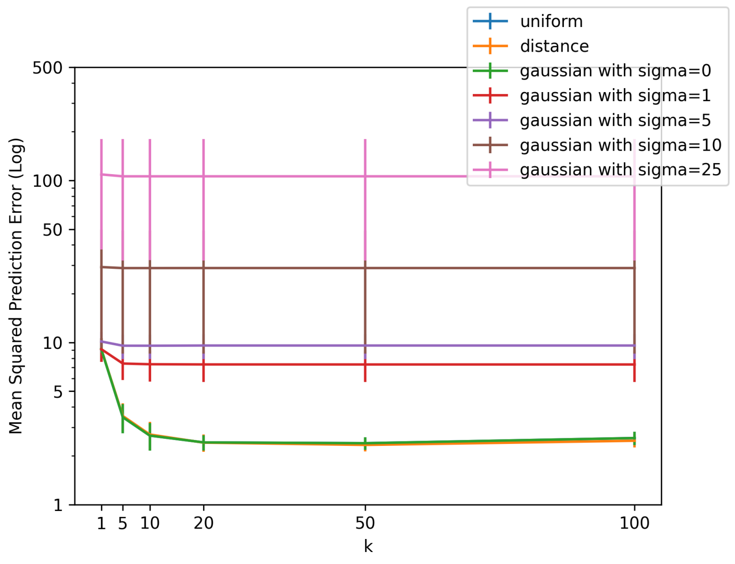
However, this too resulted in no noticeable change to our results. I suspect that the reason for this is that our other nodes combined already do a good job of capturing the different behaviour between days.

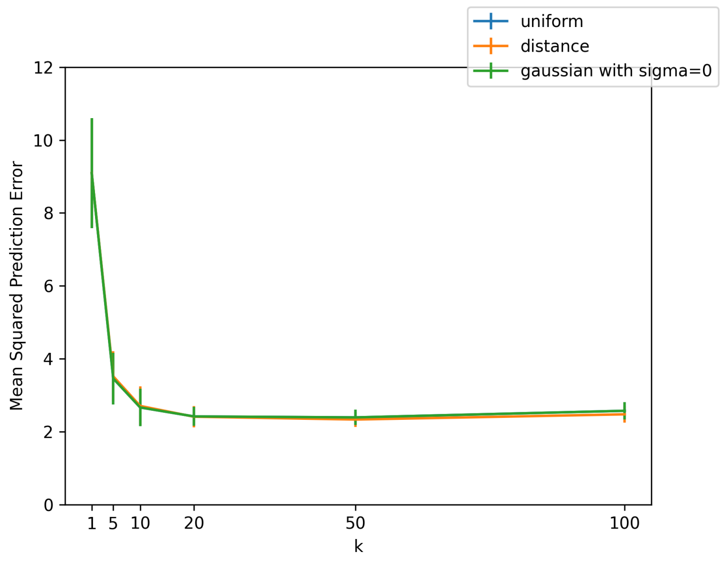
I also went back and applied this feature engineering to the previous linear regression models. They too saw no benefit from this technique. Because of these results going forwards I will not be modelling weekdays/ weekends with unique features.

Cross Validation

Number of Neighbours (K) & Distance Metric

Now with our features and dataset decided upon I went about cross-validating both our value for k, and our distance metric. For our distance metrics I tested uniform weighting, distance (Euclidean) weighting, and a series of gaussian weightings with various values of γ. I then calculated the MSPE of models trained using each of these distance metrics combined with multiple values for k, and got the following results:



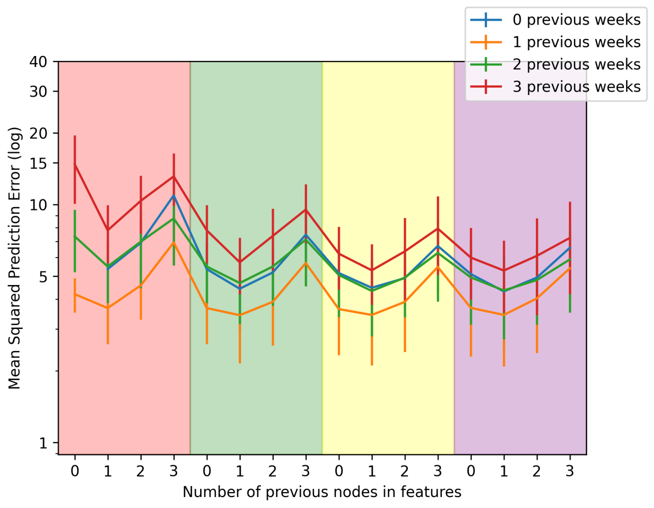


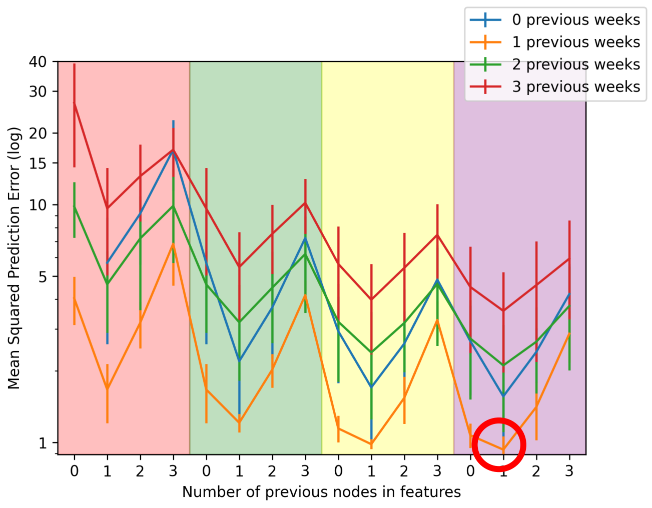
Left: MSPE with all weighting techniques Right: MSPE for best performing weighting techniques

From these graphs we can see that uniform, distance or gaussian with sigma=0 weighting and k>20 are all good choices, so the end model I decided upon used **distance weighting** with **k=50**

K-Nearest Neighbours – Convention Centre

I repeated the above feature engineering and cross validation to create a k-NN model for the convention centre bike station.





Left: Feedforward k-NN Regression (Logarithmic)

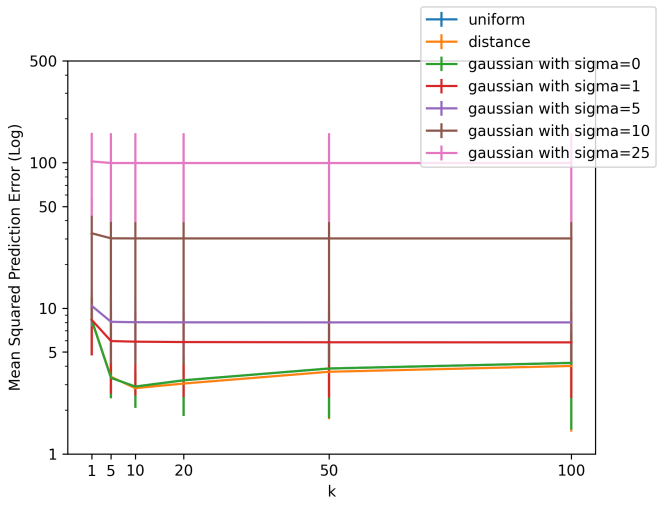
Right: Feedback k-NN Regression (Logarithmic)

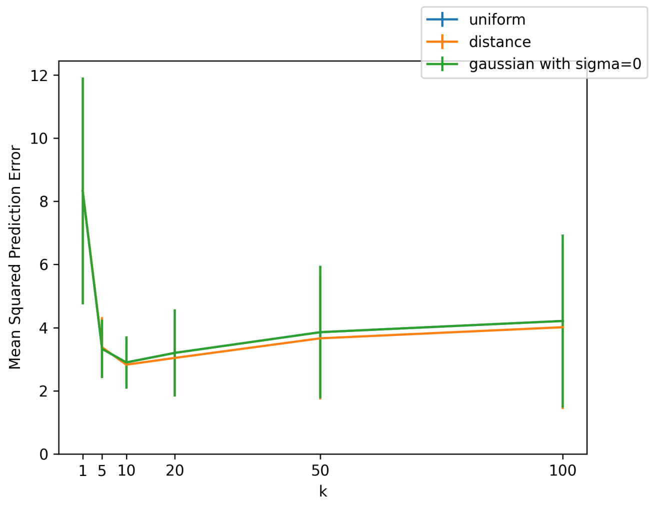
Each Zone represents how many previous days, y (0, 1, 2 or 3) are included in the feature set

0 nodes, 0 days and 0 weeks is not included since it's MSPE is so large it makes these graphs unreadable

Like before the feedback model shows a significant improvement to the performance of the best feature sets, however it doesn't make improvements across the board. Since we're only using one set of features ultimately, I choose the **feedback model** again. I also decided to use the feature set with **1 previous node, 3 previous days & 1 previous week**.

I then cross validated the value for k and the distance metric:





Left: MSPE with all weighting techniques

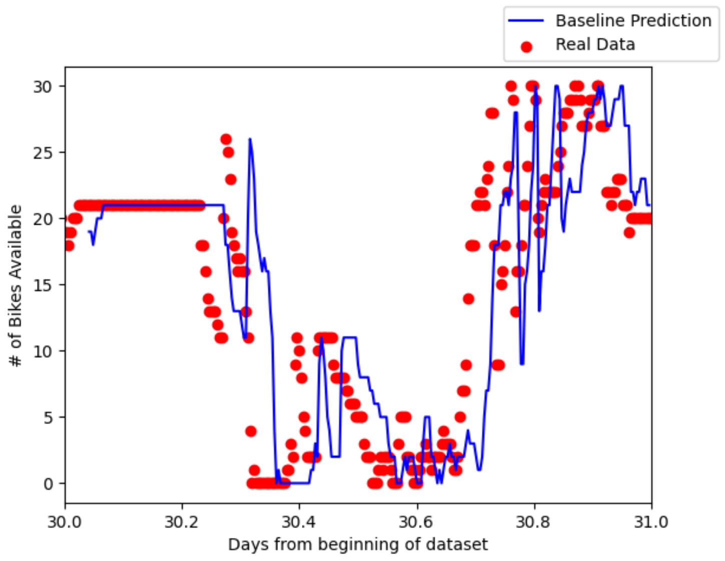
Right: MSPE for best performing weighting techniques

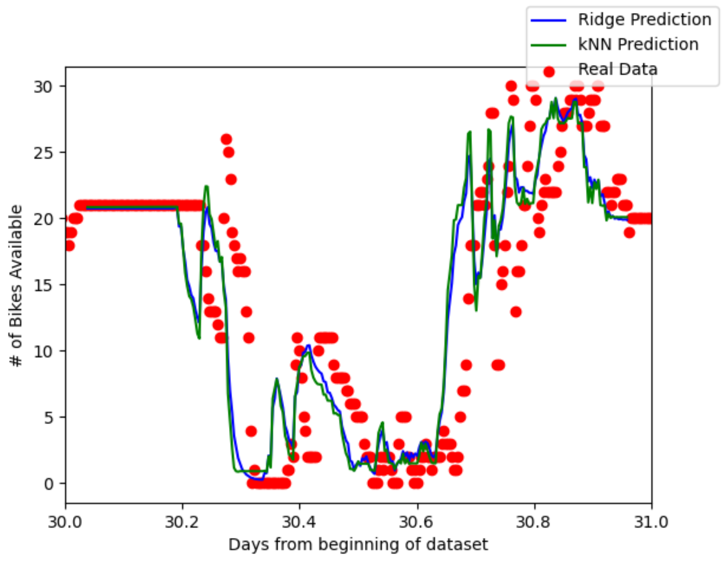
Just like last time, uniform, distance and gaussian with sigma = 0 weighting are all good choices, but this time **k=10** is the best choice for k. I also decided to use **distance** weighting since it performed marginally better at k=10 than the other options.

Evaluation

Baseline Model

In order to determine whether or not our models have found a meaningful solution for this problem I decided to compare them to a baseline model. From looking at the data it seemed to me that generally the number of bikes available doesn't change too much minute-to-minute, so I decided to make my baseline model always predicts the current value as the future output. I then plotted it's predictions, alongside the predictions of the other models for comparison.





Left: Baseline prediction for portobello harbour one hour ahead

Right: Ridge & k-NN predictions for portobello harbour one hour ahead

Data from the 4th of March 2020, with models trained using rest of dataset

For each of these models I decided to calculate three separate values in order to evaluate their performances. I calculated the Mean Absolute Error (MAE), Mean Square Error (MSE), and R2 Score, each to highlight different aspects of the models performance. MSE will punish predictions that are farther from the truth much more heavily than closer answers, whilst MAE is more forgiving towards large margins of error. Meanwhile since our baseline predictor isn't always predicting a constant value, the R2 score will give us a good sense of how our models would compare to that kind of baseline performance. I am most interested in the MSE results since I believe that is the best indicator of how well each of these models are performing but I think the other two metrics could give us some valuable insights as well.

Evaluation Results – 10 Minutes ahead

Portobello Harbour	Mean Absolute Error	Mean Square Error	R2 Score
Baseline	1.07191	5.07149	0.93983
Ridge	0.57857	1.04090	0.98559
k-NN	0.91519	1.79968	0.97357

Convention Centre	Mean Absolute Error	Mean Square Error	R2 Score
Baseline	0.54415	2.00767	0.98027
Ridge	0.26996	0.31171	0.99509
k-NN	1.43195	3.94649	0.91362

Evaluation Results – 30 Minutes ahead

Portobello Harbour	Mean Absolute Error	Mean Square Error	R2 Score
Baseline	2.20678	14.9880	0.82204
Ridge	1.48737	6.35441	0.91169
k-NN	2.00024	10.0708	0.86402

Convention Centre	Mean Absolute Error	Mean Square Error	R2 Score
Baseline	1.24509	8.69344	0.91461
Ridge	0.78494	3.07829	0.96313
k-NN	1.97092	9.49347	0.86408

Evaluation Results – 1 Hour ahead

Portobello Harbour	Mean Absolute Error	Mean Square Error	R2 Score
Baseline	3.17180	26.5701	0.68416
Ridge	2.61550	17.4915	0.74832
k-NN	3.11830	23.0597	0.68729

Convention Centre	Mean Absolute Error	Mean Square Error	R2 Score
Baseline	2.08019	21.3728	0.79020
Ridge	1.66875	13.4859	0.85547
k-NN	2.73518	23.6183	0.75262

Green = Beats baseline model in this metric Red = Beaten by baseline model in this metric

We can see a few interesting things from this data. First of all this baseline model performs a lot better than most typical baseline models (and far better than a constant value predictor according to our R2 scores), making this a difficult solution to beat. Across all of the times, both of our models were able to beat it when evaluating Portobello Harbour, however only the linear regression model is able to outperform it on the Convention Centre station. However even on Portobello Harbour, the linear regression model still beats the k-NN model, meaning that for both stations and when predicting any amount of time into the future, it is the best model available.

I also decided to take a look at the coefficients of each of the features in our linear model, and found the following results:

Station	Timestamp	Last Node	Last Week	2nd Last Week	3rd Last Week
Portobello	0.0006275	0.48995001	0.48995001	0.00916011	0.0017889
Convention	-0.00012349	0.497593290	0.497593290	0.00076569	0.00083965

Magnitude of parameter for each feature in linear regression models per dataset

Darker green = larger weighting of feature in model

From this data we can see that the most previous node and most previous week make up the vast majority of the models weighting in its predictions. The timestamp in particular is barely used at all, which makes sense since it shouldn't really correlate with an output at all in any way the previous node and previous week don't already capture. The model could possibly have been improved by completely removing the timestamp feature, although it's weighted so slightly that it probably wouldn't have changed our results too much.

Returning to the overall evaluation results, as the time we are predicting ahead increases, our predictions generally become less accurate. However even predicting an hour ahead, our linear regression model is still accurate enough to make useful predictions about how many bikes will be available. From looking at the graphs above as well it is clear that at all times throughout the day our models are generally close enough to the real values, to a degree that could actually be useful to an end user interested in these predictions.

Appendix

Dataset Plots

```
def plot_bike_availability(x, y, xlabel, ylabel, xticks, xticklabels, ax):
    ax.scatter(x, y, color='red', marker='.', s=4)
    ax.set_xticks(xticks)
    ax.set_xticklabels(xticklabels)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)


def plot_bike_availability_graphs():
    fig, axs = plt.subplots(3, 2, figsize=(10, 10))
    plt.rcParams['figure.dpi'] = 600
    # Plot entire range
    t, y = get_data_between_dates(34, "01-01-2020", "01-04-2020")
    plot_bike_availability(t, y, "Date", "# of Bikes Available", [0, 31, 60, 91],
                           ["Jan 1", "Feb 1", "Mar 1", "Apr 1"], axs[0, 0])

    # Plot range to use in program
    t, y = get_data_between_dates(34, "03-02-2020", "29-03-2020")
    plot_bike_availability(t, y, "Date", "# of Bikes Available", [0, 10, 20, 30,
40, 50],
                           ["Feb 3", "Feb 13", "Feb 23", "Mar 4", "Mar 14", "Mar
24"], axs[2, 0])

    # Plot data for one week
    t, y = get_data_between_dates(34, "03-02-2020", "10-02-2020")
    plot_bike_availability(t, y, "Date", "# of Bikes Available", [0, 1, 2, 3, 4, 5,
6],
                           ["Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun"],
axs[1, 0])
    # Plot entire range
    t, y = get_data_between_dates(65, "01-01-2020", "01-04-2020")
    plot_bike_availability(t, y, "Date", "# of Bikes Available", [0, 31, 60, 91],
                           ["Jan 1", "Feb 1", "Mar 1", "Apr 1"], axs[0, 1])

    # Plot range to use in program
    t, y = get_data_between_dates(65, "03-02-2020", "29-03-2020")
    plot_bike_availability(t, y, "Date", "# of Bikes Available", [0, 10, 20, 30,
40, 50],
                           ["Feb 3", "Feb 13", "Feb 23", "Mar 4", "Mar 14", "Mar
24"], axs[2, 1])

    # Plot data for one week
    t, y = get_data_between_dates(65, "03-02-2020", "10-02-2020")
    plot_bike_availability(t, y, "Date", "# of Bikes Available", [0, 1, 2, 3, 4, 5,
6],
                           ["Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun"],
axs[1, 1])

    axs[0, 0].annotate("Portobello Harbour", xy=(0.5, 1.05), xycoords='axes
fraction', size=24, ha='center', va='baseline')
    axs[0, 1].annotate("Convention Centre", xy=(0.5, 1.05), xycoords='axes
fraction', size=24, ha='center', va='baseline')
    axs[0, 0].annotate("Entire Range", xy=(-0.125, 0.5), xycoords='axes fraction',
size=24, rotation=90, ha='right', va='center')
    axs[1, 0].annotate("One Week", xy=(-0.125, 0.5), xycoords='axes fraction',
size=24, rotation=90, ha='right', va='center')
    axs[2, 0].annotate("Usable Data", xy=(-0.125, 0.5), xycoords='axes fraction',
size=24, rotation=90, ha='right', va='center')
    fig.tight_layout()
    fig.show()
```

Feature Engineering

```
def plot_feature_prediction_error(xaxis, mean, error, x, ax):
    if x == 0:
        ax.errorbar(xaxis[1:], mean[1:], error[1:], label="0 previous weeks")
    else:
        ax.errorbar(xaxis, mean, error, label="{} previous weeks".format(x))

def plot_knn_feature_engineering(t, y, feedforward=False, specify_weekends=0):
    for x in [0,1,2,3]:
        print("Generating error bar for knn with %i week's backtracking" % x)
        # Nodes to go back and days to go back
        backsizes = [[0, 0], [0, 1], [0, 2], [0, 3], [1, 0], [1, 1], [1, 2], [1,
3],
                     [2, 0], [2, 1], [2, 2], [2, 3], [3, 0], [3, 1], [3, 2], [3,
3]]
        for i in backsizes:
            i.append(x)
        backsize_mpse = [knn.knn_feature_engineering(t, y, 2, backsize,
specify_weekends=specify_weekends, feedforward=feedforward) for backsize in
backsizes]
        backsize_mean = [np.mean(x) for x in backsize_mpse]
        backsize_std = [np.std(x) for x in backsize_mpse]
        plot_feature_prediction_error(range(len(backsizes)), backsize_mean,
backsize_std, x, ax)


def plot_linear_feature_engineering(t, y, feedforward=False, specify_weekends=0):
    for x in [0, 1, 2, 3]:
        print("Generating error bar for linear with %i week's backtracking" % x)
        # Nodes to go back and days to go back
        backsizes = [[0, 0], [0, 1], [0, 2], [0, 3], [1, 0], [1, 1], [1, 2], [1,
3],
                     [2, 0], [2, 1], [2, 2], [2, 3], [3, 0], [3, 1], [3, 2], [3,
3]]
        for i in backsizes:
            i.append(x)
        backsize_mpse = [linear.linear_feature_engineering(t, y, 2, backsize,
specify_weekends=specify_weekends, feedforward=feedforward) for backsize in
backsizes]
        backsize_mean = [np.mean(x) for x in backsize_mpse]
        backsize_std = [np.std(x) for x in backsize_mpse]
        plot_feature_prediction_error(range(len(backsizes)), backsize_mean,
backsize_std, x, ax)
```

Cross Validation

```
def plot_knn_cross_validation(t, y):
    ks = [1, 5, 10, 20, 50, 100]
    for weight in ["uniform", "distance", gaussian_kernel_0, gaussian_kernel_1,
gaussian_kernel_5,
                   gaussian_kernel_10, gaussian_kernel_25]:
        k_mpse = [knn.knn_cross_validate(t, y, k, weight) for k in ks]
        k_mean = [np.mean(x) for x in k_mpse]
        k_std = [np.std(x) for x in k_mpse]
        ax.errorbar(ks, k_mean, k_std)


def plot_linear_cross_validation(t, y):
    cs = [0.1, 0.5, 1, 10, 100, 1000]
    for regularisation in ["l1", "l2"]:
        c_mpse = [linear.linear_cross_validate(t, y, c, regularisation) for c in
cs]
        c_mean = [np.mean(x) for x in c_mpse]
        c_std = [np.std(x) for x in c_mpse]
        ax.errorbar(cs, c_mean, c_std)
```

Evaluation

```
def make_dummy_predictions(x, y, q):
    return x[q:], y[:-q]


def eval_models(q, station):
    kf = KFold(n_splits=5)
    results = [[], [], [], [], [], []]
    t, y = get_data_between_dates(station, "03-02-2020", "29-03-2020")
    t2, y2 = get_feature_engineered_data(t, y, 1, 0, 3, specify_weekends=0)
    linear_model = Ridge(alpha=1/2)
    if station == 34:
        t3, y3 = get_feature_engineered_data(t, y, 1, 1, 1, specify_weekends=0)
        knn_model = KNeighborsRegressor(n_neighbors=50, weights="distance")
    else:
        t3, y3 = get_feature_engineered_data(t, y, 1, 3, 1, specify_weekends=0)
        knn_model = KNeighborsRegressor(n_neighbors=10, weights="distance")

    print("Evaluating Station %i" % station)
    for train, test in kf.split(t2):
        xpred, yreal, ypred = predict_q_nodes_ahead(t2[test], y2[test], q,
t2[train], y2[train], linear_model)
        results[0].append(r2_score(yreal, ypred));
results[1].append(mean_squared_error(yreal, ypred));
results[2].append(mean_absolute_error(yreal, ypred))
    for train, test in kf.split(t3):
        xpred2, yreal2, ypred2 = predict_q_nodes_ahead(t3[test], y3[test], q,
t3[train], y3[train], knn_model)
        results[3].append(r2_score(yreal2, ypred2));
results[4].append(mean_squared_error(yreal2, ypred2));
results[5].append(mean_absolute_error(yreal2, ypred2))

    print("Linear mean R2 Score: ", np.mean(results[0])); print("Linear mean MSPE
Score: ", np.mean(results[1])); print("Linear mean MAE Score: ",
np.mean(results[2]))
    print("kNN mean R2 Score: ", np.mean(results[3])); print("kNN mean MSPE Score:
", np.mean(results[4])); print("kNN mean MAE Score: ", np.mean(results[5]))

    xpred, ypred = make_dummy_predictions(t2, y2, q)
    print("Dummy R2 Score: ", r2_score(y2[q:], ypred)); print("Dummy MSPE Score: ",
mean_squared_error(y2[q:], ypred)); print("Dummy MAE Score: ",
mean_absolute_error(y2[q:], ypred))
```