Using Incremental SVD to Predict Wind Speeds

Amina Shikhalieva and Elena Peot
(as2642 and ekp38) @cornell.edu

Annotated Checklist

1. How does our project relate to the linear algebra part of CS 3220?
    a. We utilize SVD and matrix properties to implement an efficient algorithm.
2. How does our project relate to the statistics part of CS 3220?
    a. We utilize statistics to compare the estimation of winds from the test set to the actual winds, producing an estimate of the average error of the model we developed.
3. How does our project relate to the optimization part of CS 3220?
    a. We optimized the window size and approximation size k as our hyperparameter tuning process.
4. What is the application or the problem that we are solving/exploring?
    a. The primary application we are exploring is wind forecasting. However, the technique can be used beyond meteorological phenomena prediction. The technique itself allows any parameter to be estimated based on a sliding window of numerical datapoints. So it could be used to predict in the short term anything from data usage to power consumption.
5. What is the goal and how are we evaluating this goal?
    a. Our goal is to accurately and efficiently predict wind speed and direction given past data about the weather. We are evaluating our success in this goal on how well our model performs on our testing set.
6. What is the theory, relevant to our application/problem, that we will discuss?
    a. We discuss the theory of the FAST algorithm, and the advantages of the incremental SVD approach to prediction as opposed to a whole matrix SVD.
7. What have we implemented?
    a. We have implemented the FAST algorithm as a predictive means for wind speed in Houston, as well as the optimization for the results from that algorithm.
8. What are the sources of our data? (Or state that you have created synthetic data.)
    a. The data we used is obtained from the following link. Our code demonstrates how we cleaned and preprocessed it.
    b. https://www.kaggle.com/selfishgene/historical-hourly-weather-data?select=weather_description.csv

Introduction

In this project we use the Fast Approximate Subspace Tracking (FAST) algorithm to forecast changes in wind speed in a particular location, given prior data about the local weather conditions. To simplify, we will focus solely on the magnitude, and not the direction, of the winds. With this incremental singular value decomposition (SVD) technique, we will make singular-value prediction more efficient and feasible at larger scales. We will also experiment with hyperparameters to study the behavior of FAST and how it deals with different types of input data.

Besides weather prediction, the incremental SVD has been studied for movie recommendation systems [2], and in computer vision [3]. It is a powerful tool in making prediction more accessible and computationally feasible by eliminating the need to repeatedly take the SVD of a usually large, but iteratively tweaked, matrix.

Procedure

The power of the incremental SVD is in the efficiency of calculation--instead of recalculating the SVD of the entire matrix every time we get new data, we can calculate the SVD of a smaller matrix, and do some calculations to approximate the SVD of our large, updated matrix. (Note: for our purposes, this does not make a huge difference, since our data only has dimension 5, but in real-life applications, this cuts down very significantly on computational power.) See Code Appendix [1,2] for implementation details. Our procedure is as follows. First we construct a $n * c$ matrix M, where c is the window size (number of time instances stored in the matrix) and m is the dimension of the data vector. For us, $n = 5$, with the data being humidity, pressure, temperature, wind speed, and wind direction. For instance, if m_0 is the first column of M, it is an $n * 1$ column vector, where each value is a different property at time 0.

We then obtain the k-truncated SVD using a built-in numpy method:

$$M \approx U_k \Sigma_k V_k^T$$

We then "slide" the window of the matrix to the right upon receiving new data, so:

$$M_{new} = \left[M_{old}[:,\ 2:] \quad m_{c+1}\right]$$

Now we calculate a low-rank approximation A to M_{new}.

$$A = U_{,k}^{T} M_{new}$$

We can rewrite A as:

$$A = [U_k \ \ q]\, E\ ,$$

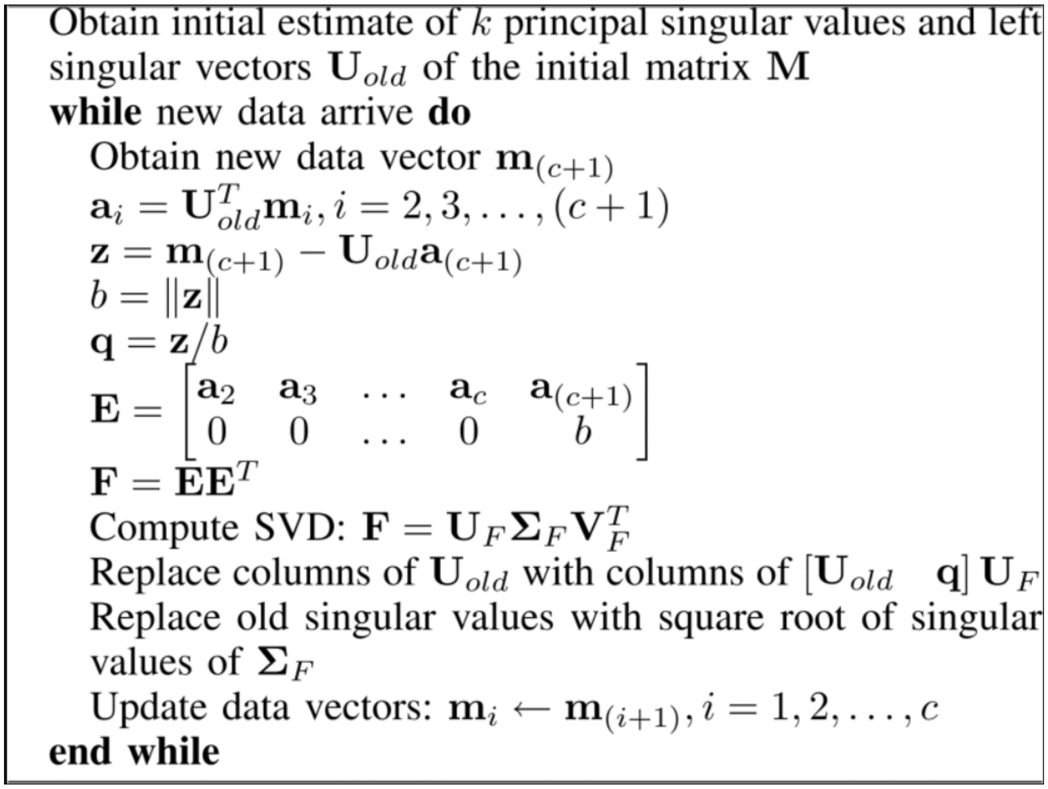
where $q = \frac{m_{c+1} - U_{old} a_{c+1}}{||m_{c+1} - U_{old} a_{c+1}||}$ and $E = \begin{bmatrix} a_2 & a_3 & \dots & a_c & a_{(c+1)} \\ 0 & 0 & \dots & 0 & b \end{bmatrix}$

Now we define a new matrix F:

$$F = EE^{T}$$

Since F is a size $(k + 1) * (k + 1)$ matrix, it is easier to factorize into the SVD. The square root of the singular values of F equal the singular values of A. These singular values are what we use to predict the behavior of the wind. While they are not a prediction for the value of the magnitude of the wind itself, they should increase and decrease with the wind, allowing us to get an idea of how the wind will change in magnitude in the short term future.

We used the following algorithm from [1] to write our code for the FAST procedure [Appendix, 1,2]:

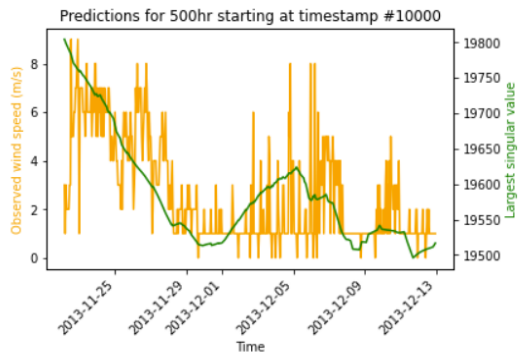


```
Obtain initial estimate of $k$ principal singular values and left
singular vectors $\mathbf{U}_{old}$ of the initial matrix $\mathbf{M}$
while new data arrive do
    Obtain new data vector $\mathbf{m}_{(c+1)}$
    $\mathbf{a}_i = \mathbf{U}_{old}^T \mathbf{m}_i, i = 2, 3, \ldots, (c+1)$
    $\mathbf{z} = \mathbf{m}_{(c+1)} - \mathbf{U}_{old}\mathbf{a}_{(c+1)}$
    $b = \|\mathbf{z}\|$
    $\mathbf{q} = \mathbf{z}/b$
    $\mathbf{E} = \begin{bmatrix} \mathbf{a}_2 & \mathbf{a}_3 & \ldots & \mathbf{a}_c & \mathbf{a}_{(c+1)} \\ 0 & 0 & \ldots & 0 & b \end{bmatrix}$
    $\mathbf{F} = \mathbf{E}\mathbf{E}^T$
    Compute SVD: $\mathbf{F} = \mathbf{U}_F \mathbf{\Sigma}_F \mathbf{V}_F^T$
    Replace columns of $\mathbf{U}_{old}$ with columns of $[\mathbf{U}_{old} \quad \mathbf{q}]\,\mathbf{U}_F$
    Replace old singular values with square root of singular
    values of $\mathbf{\Sigma}_F$
    Update data vectors: $\mathbf{m}_i \leftarrow \mathbf{m}_{(i+1)}, i = 1, 2, \ldots, c$
end while
```

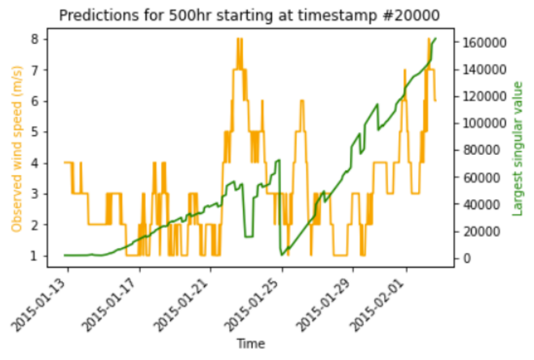
We decided to focus on Houston out of the 37 cities we have data for because it has the fewest NaN values. NaN values still posed a problem, however, and as can be seen in the hyperparameter optimization stage later [Appendix, 4]. For simplicity and because there were relatively few NaN values, we chose to skip over chunks of data where they were present.

Preliminary Results

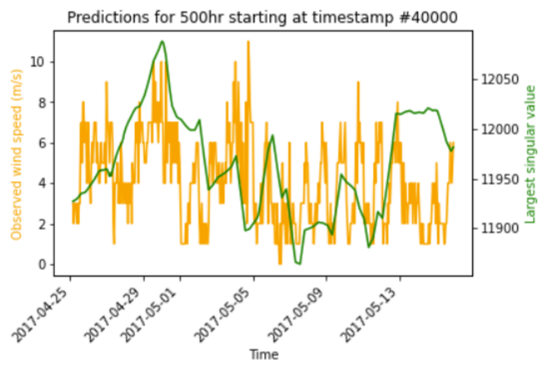
Below are some preliminary plots to help visualize the behavior of the predictions. The timeframe for each plot is given. We optimized the window size and k value for each timeframe, and also show the mean squared error for the predictions (more on this later). The results are somewhat ambiguous in quality. In particular, in earlier time frames (time frames of the same length, but that start earlier), we tend to get visually better fits, and lower MSE values to accompany. We do not know why this could be, considering all the models "trained" for the same length of time. A possible guess as to why this may be could be that we assume wind patterns do not change in predictability over years. Perhaps seasonal or annual climate patterns did change over time, or even the instrumentation with which the data was collected. Or, maybe our results were just a product of chance. In any case, more interval samples need to be taken to reach a hypothesis.



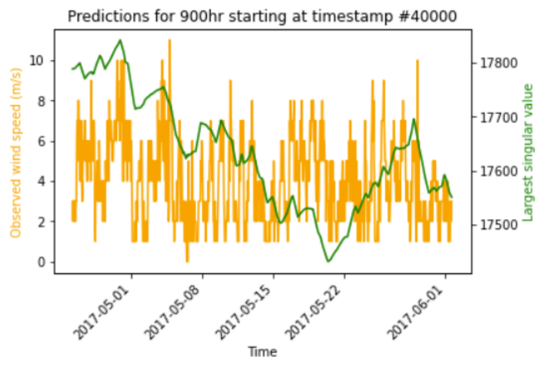
```
MSE: 3.1856521170016103
```



```
MSE: 3.8498806238070933
```



```
MSE: 9.045918871172027
```



```
MSE: 10.856249696455471
```

Hyperparameter Optimization and MSE

The way we optimized our model was by breaking our data for one city (Houston) down into 500-hr intervals. There are two hyperparameters: the approximation size k, and the size of the "window", or the width of the matrix we factorize.

We tested the accuracy of the fit using the mean squared error (MSE) of our sigma values and the observed wind speeds. We first scaled the values, since the largest singular values are usually on the order of ten thousand, while the wind measurements were generally under 15 m/s:

$$\hat{s} = \frac{s-min(S)}{max(S)-min(S)} * (max(W) - min(W)) + min(W),$$

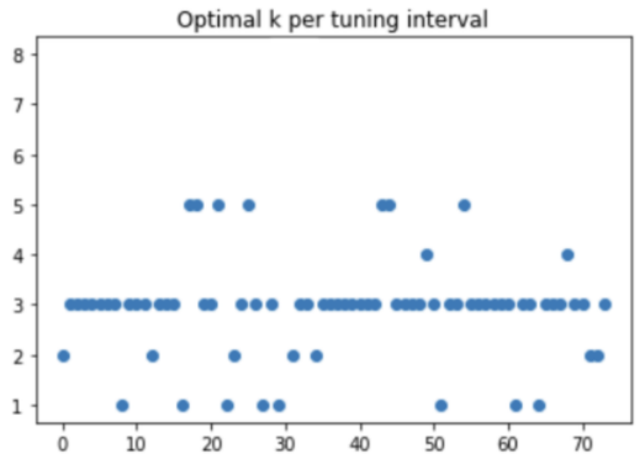
Where $\hat{s}$ is the scaled largest singular value, s is the raw largest singular value, and S and W are the sets of largest singular values and observed wind speeds across all tuning intervals, respectively. We then calculated the mean squared error of the two sets:

$$MSE = mean[(\hat{S} - W)^2]$$

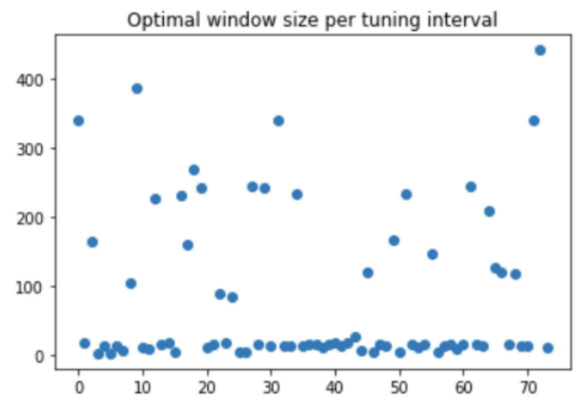
We optimize our model by selecting the hyperparameters that minimize the MSE of the largest singular values and observed wind speeds for the given time frame.

Optimization and Analysis

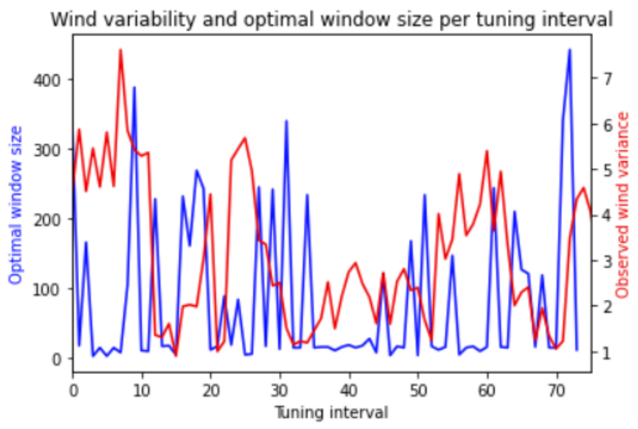
We implemented the above described hyperparameter tuning, minimizing the error with respect to the values of k (the rankof the SVD of the data matrix M) and window size.
Interestingly, our optimal k values ranged from 1 to 5, with a clear mode of 3:



In general, this means a $k = 3$ rank approximation is suitable to accurately describe our rank 5 data matrix.
Our optimal window size also varied considerably, between 3 and 442, with a mean of approximately 85. However, most values are low, less than 20, so in general, a smaller window size is sufficient, with some notable outliers:



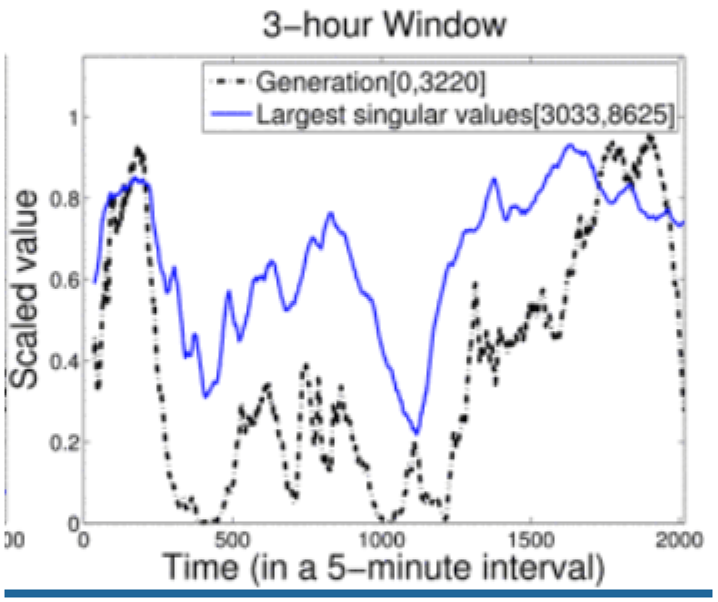
We wanted to look deeper into why this could be. So we also kept track of the variance of wind speeds at each interval that we optimized hyperparameters at. There was a slight bug in our code that overcounted the intervals for which the variance was recorded (when we skipped the interval because of NaN values, we still mistakenly recorded the wind speed variance). Unfortunately we were not able to recover the cleaned values, since our code took two hours to run and we did not want to lose our data for the sake of a relatively small number of overcounts.



While we did not do any quantitative analysis on these results because of the overcount error, to the naked eye, it seems there may have been some correlation between the variability in wind measurements and the optimal window size. Intervals in which there was more variance in observed wind speeds were better fit by matrices with larger window sizes. Intuitively, this makes sense, because a smaller window is more sensitive to the presence of outliers.With more variance, we need more sample points to be able to accurately represent the statistical measures.

Conclusion

We have concluded that our method is somewhat inaccurate. This is probably mainly due to the fact that in general, wind is hard to predict, especially in one-hour intervals. Other factors may include the low dimensionality of the data we used. We only used 5 weather measurements per time marker because of the limitations of the data we found. However, many more observations could have been made, such as potential temperature, dewpoint, etc. Also, the nature of weather prediction in general is notoriously difficult, so no matter the quality of data we use, there will likely always be high unpredictability in anything meteorological related. Furthermore, previous literature[1] also fails to produce exactly accurate predictions:

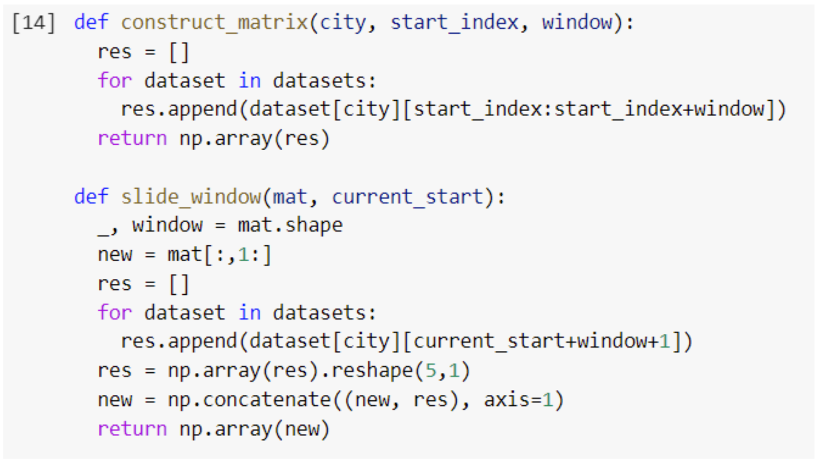


However, considering the efficiency and relatively straightforward nature of this technique, it is still remarkably useful. It is somewhat easy to always have an incoming source of metrics, and computationally, the efficiency of the incremental SVDtechnique makes this a very feasible tool to predict winds in the near future, especially considering how difficult it has been historically to be able to predict weather phenomena in general.

Further studies can look into why earlier time intervals seem to work better for predicting wind, despite having the same “training time” as instances later in time. Further studies may also take into account samples taken from several weather monitoring stations in the region.

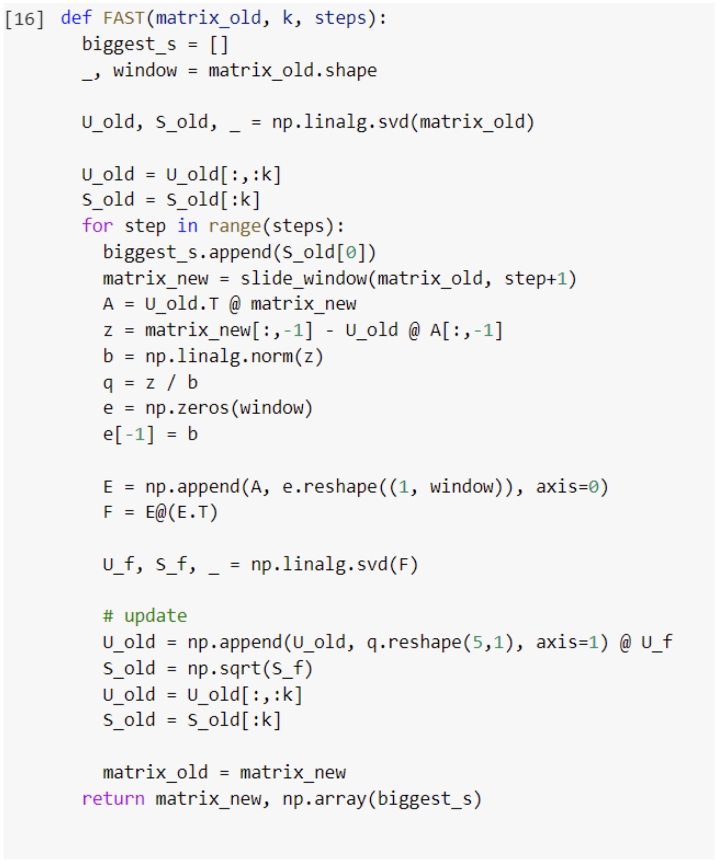
Code Appendix

Matrix operations and incremental SVD:



```
[14] def construct_matrix(city, start_index, window):
       res = []
       for dataset in datasets:
         res.append(dataset[city][start_index:start_index+window])
       return np.array(res)

     def slide_window(mat, current_start):
       _, window = mat.shape
       new = mat[:,1:]
       res = []
       for dataset in datasets:
         res.append(dataset[city][current_start+window+1])
       res = np.array(res).reshape(5,1)
       new = np.concatenate((new, res), axis=1)
       return np.array(new)
```

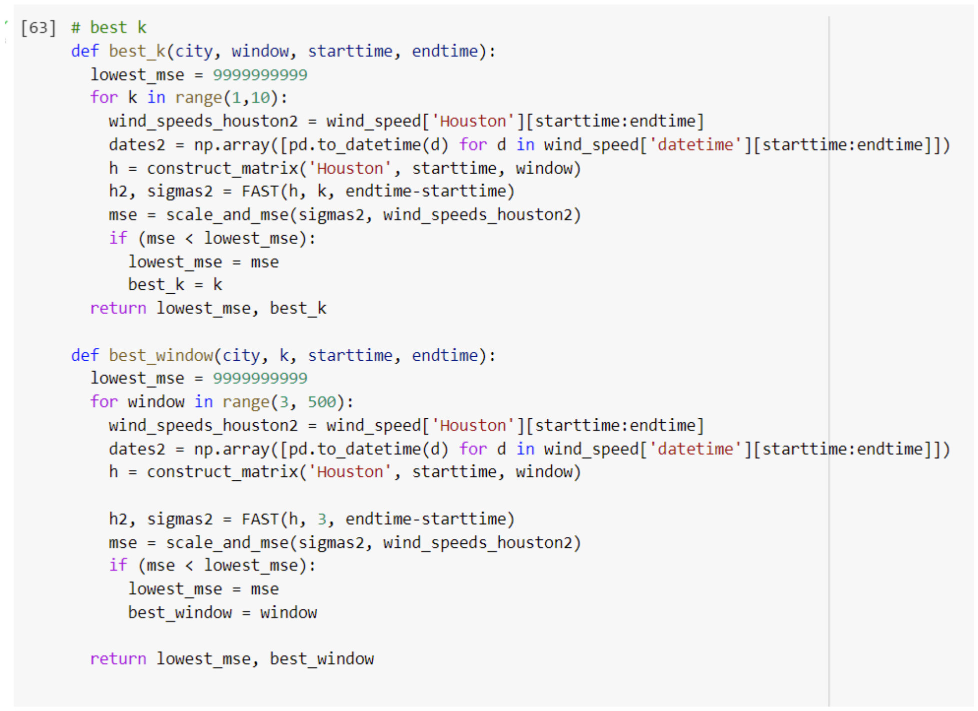
[1]



```
[16] def FAST(matrix_old, k, steps):
       biggest_s = []
       _, window = matrix_old.shape

       U_old, S_old, _ = np.linalg.svd(matrix_old)

       U_old = U_old[:,:k]
       S_old = S_old[:k]
       for step in range(steps):
         biggest_s.append(S_old[0])
         matrix_new = slide_window(matrix_old, step+1)
         A = U_old.T @ matrix_new
         z = matrix_new[:,-1] - U_old @ A[:,-1]
         b = np.linalg.norm(z)
         q = z / b
         e = np.zeros(window)
         e[-1] = b

         E = np.append(A, e.reshape((1, window)), axis=0)
         F = E@(E.T)

         U_f, S_f, _ = np.linalg.svd(F)

         # update
         U_old = np.append(U_old, q.reshape(5,1), axis=1) @ U_f
         S_old = np.sqrt(S_f)
         U_old = U_old[:,:k]
         S_old = S_old[:k]

         matrix_old = matrix_new
       return matrix_new, np.array(biggest_s)
```

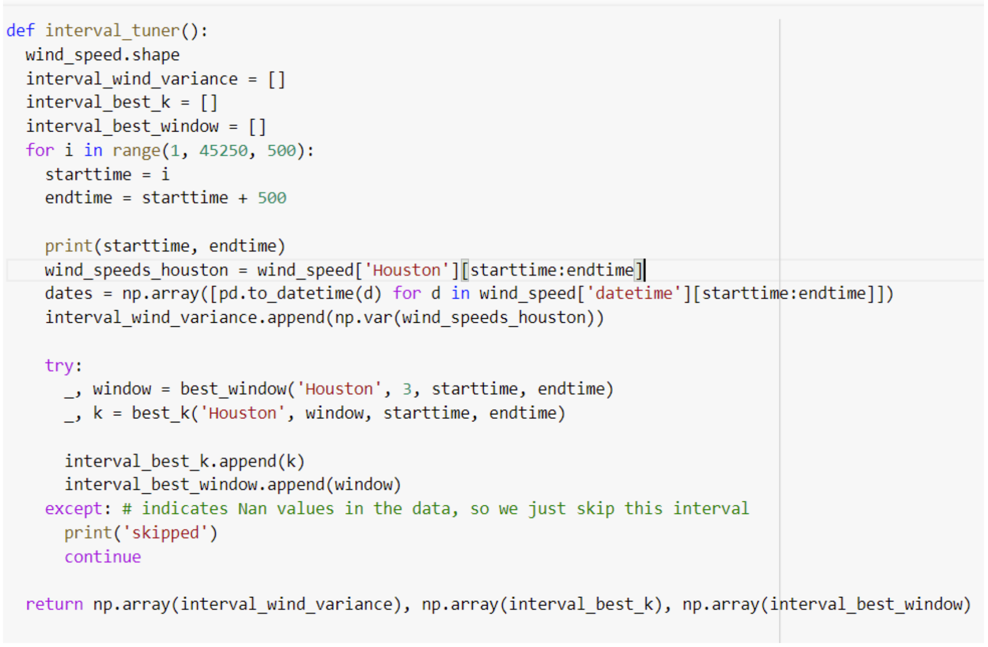
[2]

Hyperparameter optimization



```
[63] # best k
     def best_k(city, window, starttime, endtime):
       lowest_mse = 9999999999
       for k in range(1,10):
         wind_speeds_houston2 = wind_speed['Houston'][starttime:endtime]
         dates2 = np.array([pd.to_datetime(d) for d in wind_speed['datetime'][starttime:endtime]])
         h = construct_matrix('Houston', starttime, window)
         h2, sigmas2 = FAST(h, k, endtime-starttime)
         mse = scale_and_mse(sigmas2, wind_speeds_houston2)
         if (mse < lowest_mse):
           lowest_mse = mse
           best_k = k
       return lowest_mse, best_k

     def best_window(city, k, starttime, endtime):
       lowest_mse = 9999999999
       for window in range(3, 500):
         wind_speeds_houston2 = wind_speed['Houston'][starttime:endtime]
         dates2 = np.array([pd.to_datetime(d) for d in wind_speed['datetime'][starttime:endtime]])
         h = construct_matrix('Houston', starttime, window)

         h2, sigmas2 = FAST(h, 3, endtime-starttime)
         mse = scale_and_mse(sigmas2, wind_speeds_houston2)
         if (mse < lowest_mse):
           lowest_mse = mse
           best_window = window

       return lowest_mse, best_window
```

[3]



```
def interval_tuner():
  wind_speed.shape
  interval_wind_variance = []
  interval_best_k = []
  interval_best_window = []
  for i in range(1, 45250, 500):
    starttime = i
    endtime = starttime + 500

    print(starttime, endtime)
    wind_speeds_houston = wind_speed['Houston'][starttime:endtime]
    dates = np.array([pd.to_datetime(d) for d in wind_speed['datetime'][starttime:endtime]])
    interval_wind_variance.append(np.var(wind_speeds_houston))

    try:
      _, window = best_window('Houston', 3, starttime, endtime)
      _, k = best_k('Houston', window, starttime, endtime)

      interval_best_k.append(k)
      interval_best_window.append(window)
    except: # indicates Nan values in the data, so we just skip this interval
      print('skipped')
      continue

  return np.array(interval_wind_variance), np.array(interval_best_k), np.array(interval_best_window)
```

[4]

Bibliography

[1] C. Kamath and Y. J. Fan, "Incremental SVD for Insight into Wind Generation," 2014 13th International Conference on Machine Learning and Applications, 2014, pp. 441-446, doi: 10.1109/ICMLA.2014.77.

[2] Sarwar, Badrul & Karypis, George & Konstan, Joseph & Riedl, John. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. Fifth International Conference on Computer and Information Science.

[3] Tat-Jun Chin, K. Schindler and D. Suter, "Incremental kernel SVD for face recognition with image sets," 7th International Conference on Automatic Face and Gesture Recognition (FGR06), 2006, pp. 461-466, doi: 10.1109/FGR.2006.67.

Dataset:
https://www.kaggle.com/selfishgene/historical-hourly-weather-data?select=weather_description.csv