Introduction

Tesla Super Charger Network Growth Model and Decision Support System

Cole MacLean - May 13, 2016

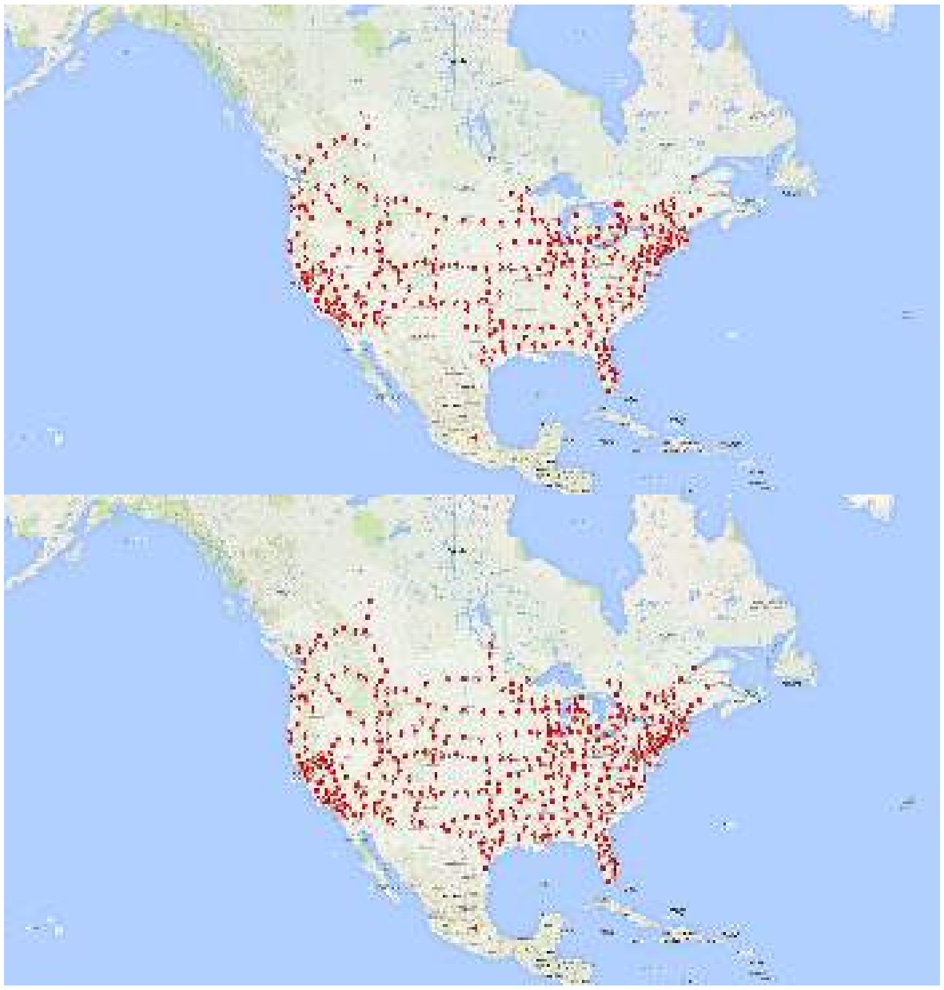
Preface

This notebook is an extension of the Exploratory Data Analysis performed on Tesla's Supercharger network data set. Using the insights and intuition developed during the EDA, an attempt at developing an intelligent network growth model is performed. The full EDA analysis can be found here. (https://lab.beakernotebook.com/#/publications/6451bcd8-096e-11e6-8b45-2b70e59230e5)

Introduction

With the recent unveiling of Tesla's Model 3 and pre-orders approaching 400,000, the internet has been buzzing with Tesla discussions and analysis. One of Tesla's key differentiators from other proposed mass market Electric Vehicals (EVs) is its Super Charger (SC) network that provides 170 miles of range in 30 minutes source. (https://www.teslamotors.com/supercharger) With Elon Musk stating plans to double the size of the SC network by the end of 2017, a large amount of planning, resources and investment are being allocated to this network expansion.

Tesla 2016 Planned North America Network Growth



source (https://www.teslamotors.com/supercharger)

Project Overview

The goal of this project is to build an intelligent decision support system to provide recommendations for where Tesla should expand their Super Charger (SC) network to maximize its effectiveness in creating a robust charging infrastructure for EVs in North America.

Project Plan

The development of the decision suppport system will require a few distinct pieces of the puzzle to build an adequate model of the decision process performed in extending the Tesla Super Charger network. These steps are listed and briefly summarized here.

1. Development of the SCNetwork object - A python class that extends the functionality of a networkx's Graph object to provide abstracted infrastructure and utility functions used in this anaysis.

2. Expansion node search space hueristics - In order to make a decision on where to expand the network to next, we first need a set of expansion possibilities. Search hueristics are built to establish and reduce the search space to produce a robust but tractable subspace of potential expansion locations that we can analyze further and ultimately pick the ideal expansion node.
3. Model Expanison City Selection - We will utilize 2 approaches in an attempt to discover the growth mechanism of the SC network:
   i. The first approach will use Classification techniques to identify appropriated expanson nodes
   ii. The second approach will use Utility Theory to find a cost function that accurately selects appropriated expanson nodes using attributes of the network
4. Model validation - Use the held out dataset of Supercharges currently under construction to validate how well the developed model matches Tesla's future SC plans
5. Bringing it all together - in order to visualize and interact with the above model, the d3.js network graphic here (http://cole-maclean.github.io/MAI-CN/) will be expanded to include extra information and interaction capabilities to allow users to better use and understand the developed network expansion model.

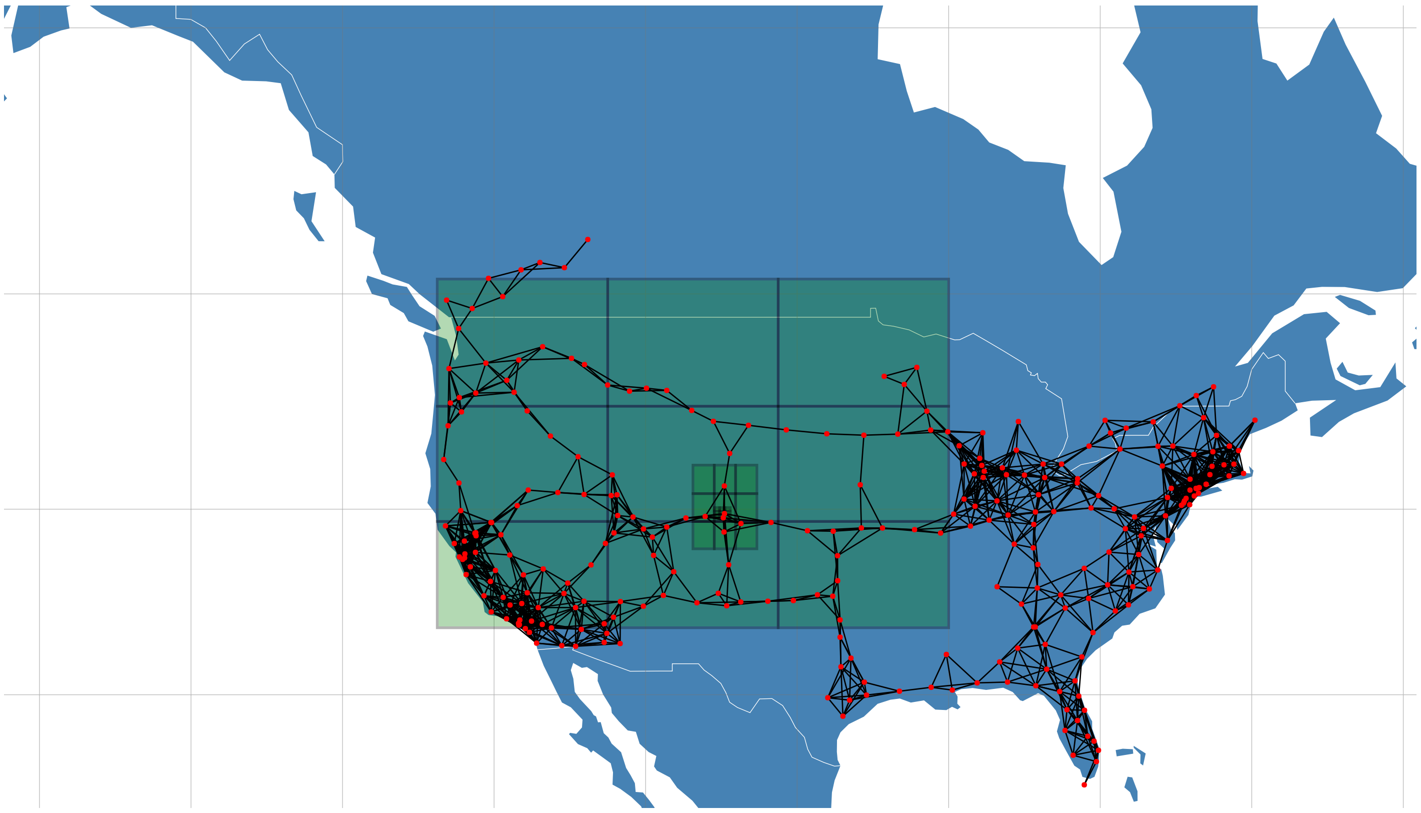
- Initial Setup

The folow scripts import the required python libraries used in this analysis, the network and population datasets and store them in the beaker namespace.

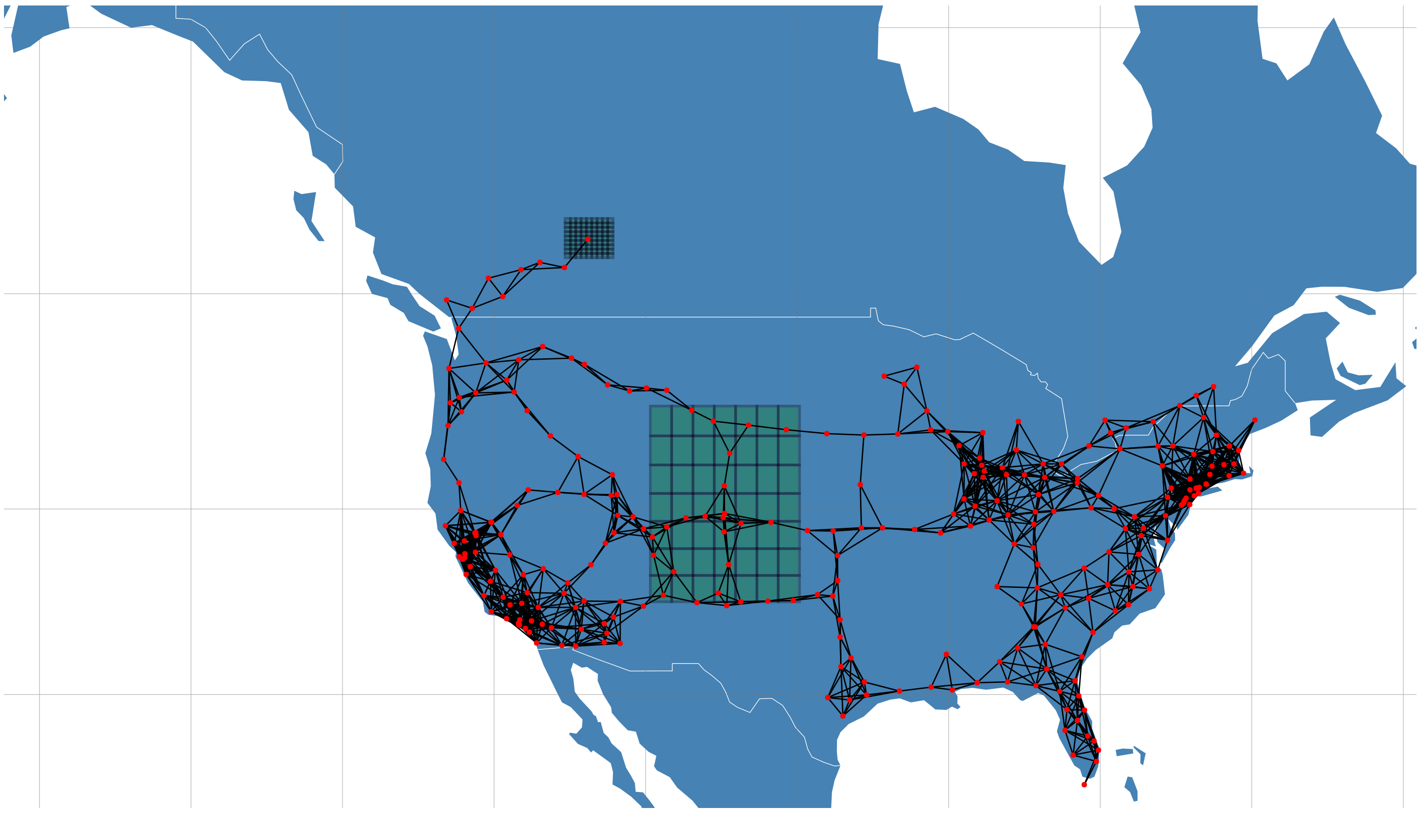
- Geohashing

Before delving into the analysis, the concept of geohashing needs to be introduced, as it is utilized extensively throughtout this analysis. Geohashing is a technique of encoding geographical locations (ie. GPS coordiantes) as strings that represent rectangle bounding boxes of customizable precision. The longer the encoded string is, the more precise the geographic bounding box. This specialized encoding and the attribute of customizable precision is very useful in identifying if 2 locations are "close" to each other. We can use this concept to efficiently search geographically for items that fall within a certain distance from one another.

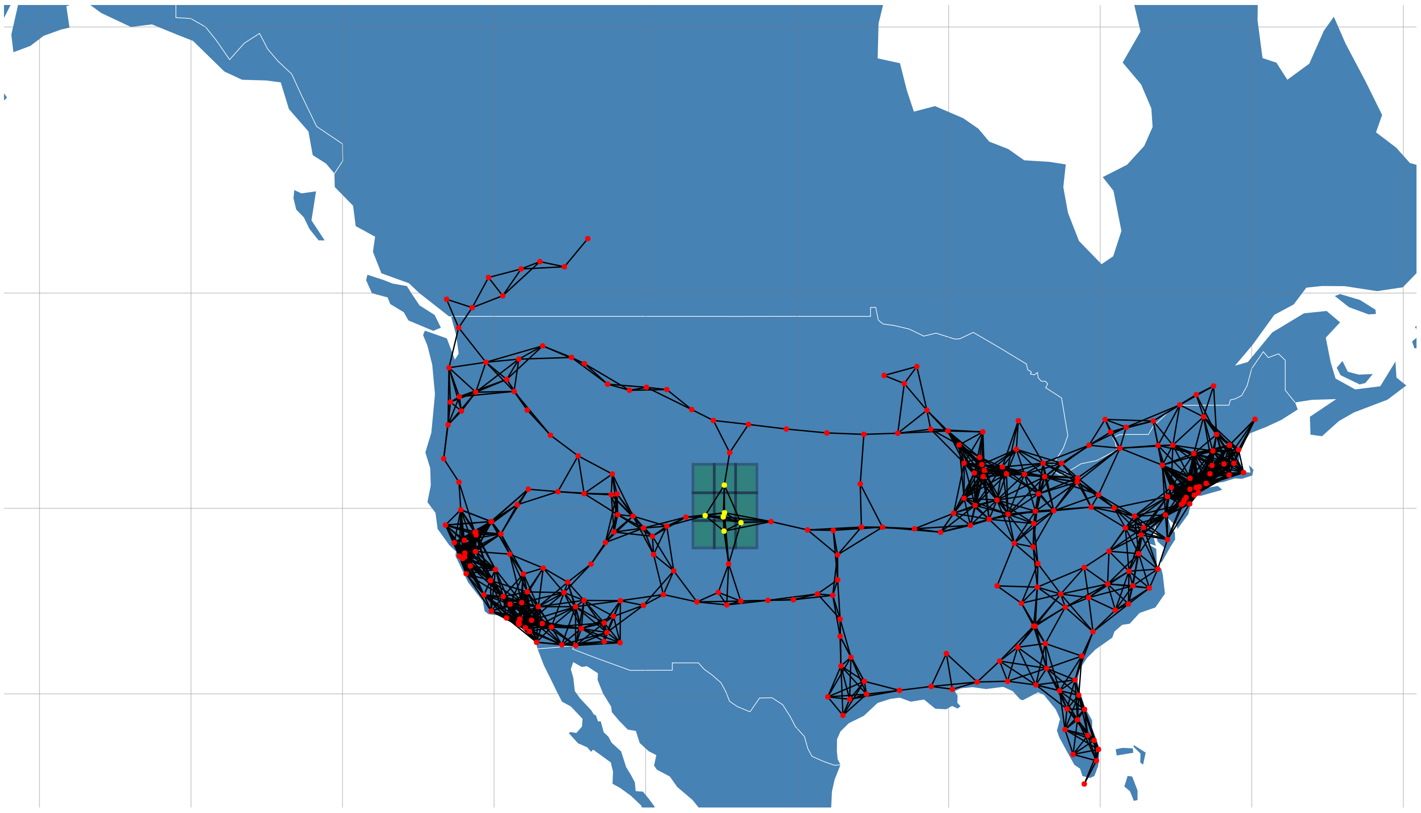
Ilustrated below are the exapanded bounded boxes for the Denver Supercharger at precisions 2,3 and 4.



One draw back to this technique is the large differences in precision between precisions 2, 3 and 4. However, using iterative expansions of any desired precision, we can create more refined bounding boxes of any size.



Using the above techniques, we can find all Supercharger that are "close" (for any arbitrary definition of close), to any other Supercharger.



With this abstraction, we can compute extremely efficiently the geographical searches across the network, and North America in general, required in this analysis.

Network Expansion Search Space

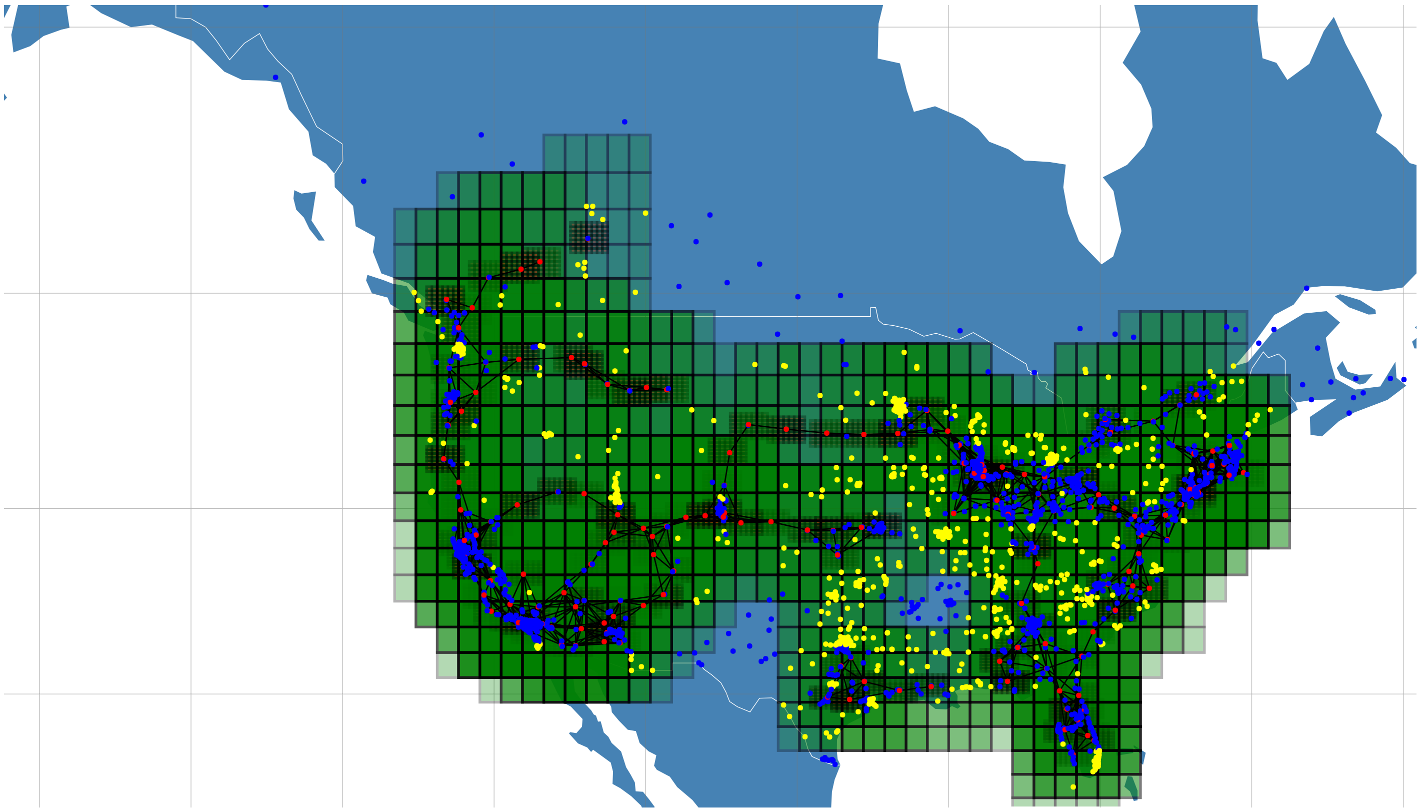
In order to simulate the expansion of the SC network, we first need to define the set of potential expansion locations. One option would be to naively utilize every city within the city-populations dataset. However, with 5,208 cities in the dataset and the need to iteratively connect each city to the network to test the utility of that connection, this option is likely computationally infeasible. As a super rough estimate, using the average network degree of 9 as the number of connections for each added city, and assuming 1 second per connection (google maps API call time), it would take 46,809 seconds to add a single node to the network. To simulate a network of 100 nodes would take approximately 4,680,900s, or 54 days. Clearly, a more intelligent approach is required.

There are 3 parameters we can use as filters to reduce the search space:

1. The first parameter is structurally inherent to the network: the max connection range between SCs (ie. 346 kms). We can use this characteristic of the network to filter-out all cities that do not exist within the maximum connection distance of any SC currently in the network. These SCs are not reachable by the network and could never be added by our definition of network connections.
2. The second parameter filters cities based on population size. Defining the parameter as Major Cities, we can exclude all cities that fall under this population threshold.
3. The last parameter uses our assumption that part of the driving force behind network expansion is to have a larger % of the availible population represented by the network. Using our definition of network penetration, we can filter-out cities that are already represented by an existing SC in the network.

Using these filters, we can obtain a computationally tractable search space. The graphic below visualizes the filtering process for the network with 180 SCs, using a max distance of 346km, a major city population threshold of 15,000 people, and a penetration grid of 4 precision expanded 3 times (about 100kmX100km). This filtering reduces the search space from 5,208 cities to 2,264 major cities, and further filters to a total of 776 potential expansion cities, a seach space reduction of 85%. The red nodes are SCs currently part of the network, the yellow nodes are potential expansion locations based on the above hueristics, and blue nodes are major cities filtered out of the search space. Using these filtering parameters eliminates 38% of the actual SCs in the network, and only makes it possible to find 62% of real SCs, but balances real-network findability with computational tractability. As a first-pass model, I think these numbers are acceptable, but should be one of the first areas of refinement for model improvement as this is likely the greatest source of error in the current model.

```
expansion search cities = 776
```



Network Modeling

With the search space algorithm defined above, we can now build a model that, given the potential expanision locations, selects the ones most similiar to the locations selected in the real SC network.

Potential Models

For our model, we will only rely on the network expansion charactertistics as predictor attributes of network expansion. This assumes that the network growth attributes can effectively discriminate correctly appropriate locations for network expansion. This is a very big assumption, and drastically limits the effectiveness of our model, but the real-world decision of where to build the next Supercharger is extremly complex and will consider things like Tesla ownship densities, favorable regulatory/ subsidy conditions and construction costs, which are outside the scope of this model. Using the network growth attributes to build the model, we can develop a system that provides recommendations of suitable locations on the basis of generating networks that maximize these desirable characteristics.

Clasification Model

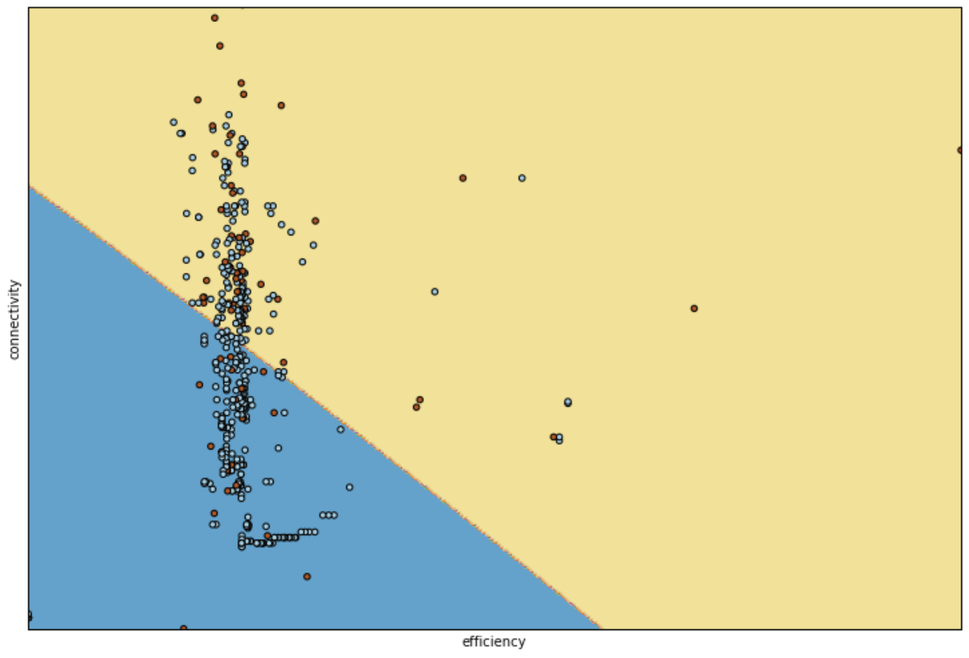
Using the actual SC network and the order that each SC was added, we can calculate the network attribute values for each SC when it was added to the network. These are our positively labelled instances in our classification problem, and all of the SCs that were availible in the search space but not selected are our negatively labelled instances. Using a 1 to represent SCs that were successfully added to the network, and 0's for locations not added, we can generate a matrix of positive and negative instances that we can plot and attempt to build a classificaiton model with.

Index	connectivity	efficiency	label

0	0	4.4700	0.0255	1.0
1	1	3.6200	0.0162	1.0
2	2	3.1100	0.0118	0.0
3	3	3.1200	0.0118	0.0
4	4	2.9100	0.0115	0.0
5	5	2.9300	0.0115	0.0
6	6	2.9300	0.0113	1.0
7	7	4.3200	0.0102	0.0
8	8	4.3200	0.0081	1.0
9				

With this data matrix, we train a simple SVM classifier with some hyperparameter optimization, to split the dataset into "good" and "bad" potential expansion locations. Once the model is trained, we can review its confusion matrix on the heldout test set, and tweaking the code from here (http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html), we can visualize the resulting classification model.

```
{'gamma': 0.001, 'class_weight': {1: 6}, 'kernel': 'rbf', 'C': 10}
[[47 26]
 [ 7 11]]
```



Reviewing the plot and corresponding confusion matrix, the results seem fairly reasonable. From our previous discussion, we know our current dataset is not comprehensive enough to fully explain the actual classification of good/bad expansion locations, but the above classification agrees with our assumption that a bias exists for greater connectivity and efficiency. Essentially, the above classification provides a network characteristic threshold to further filter out potential expansion locations. Although we lose about 40% of our positive examples using this threshold, we are able to eliminate 64% of the negative examples. The above plot also informs us that any model developed from the information we have, as currently structured, will not adequately model the actual SC network. However, as is the case with all experiements, the information gained is still useful. The above model can be used to assist decision makers as a blunt filter to reduce the subset of locations that go on to be rigorously analyzed. It is also a foundation to build upon, for more refined models that incorporated more complex features like Tesla ownership densities and measures for a cities "EV friendliness".

− Utility Model

Using utility theory, we can attempt to discover a cost function using the attributes of the network to calculate the expansion utility of a potential expansion location. We can then optimize this function by appropriately weighting each attribute such that the network generated from this function is the most similar to the real SC network. In order to build this model, we need to define a similarity measure. This score has been defined as the sum of the SCs within the bounded box of a SC in the real network, defined by a geohash precision of 4, expanded twice (ie about 60kmX60km), divided by the total number of SCs in the network.

From the EDA, we know the SC network has gone through 3 different phases of growth. In order to simplify our model, we attempt only to model the later stage of the network, as this is what will be used for predicting future network expansion. The last 74 SCs are used for the optimization.

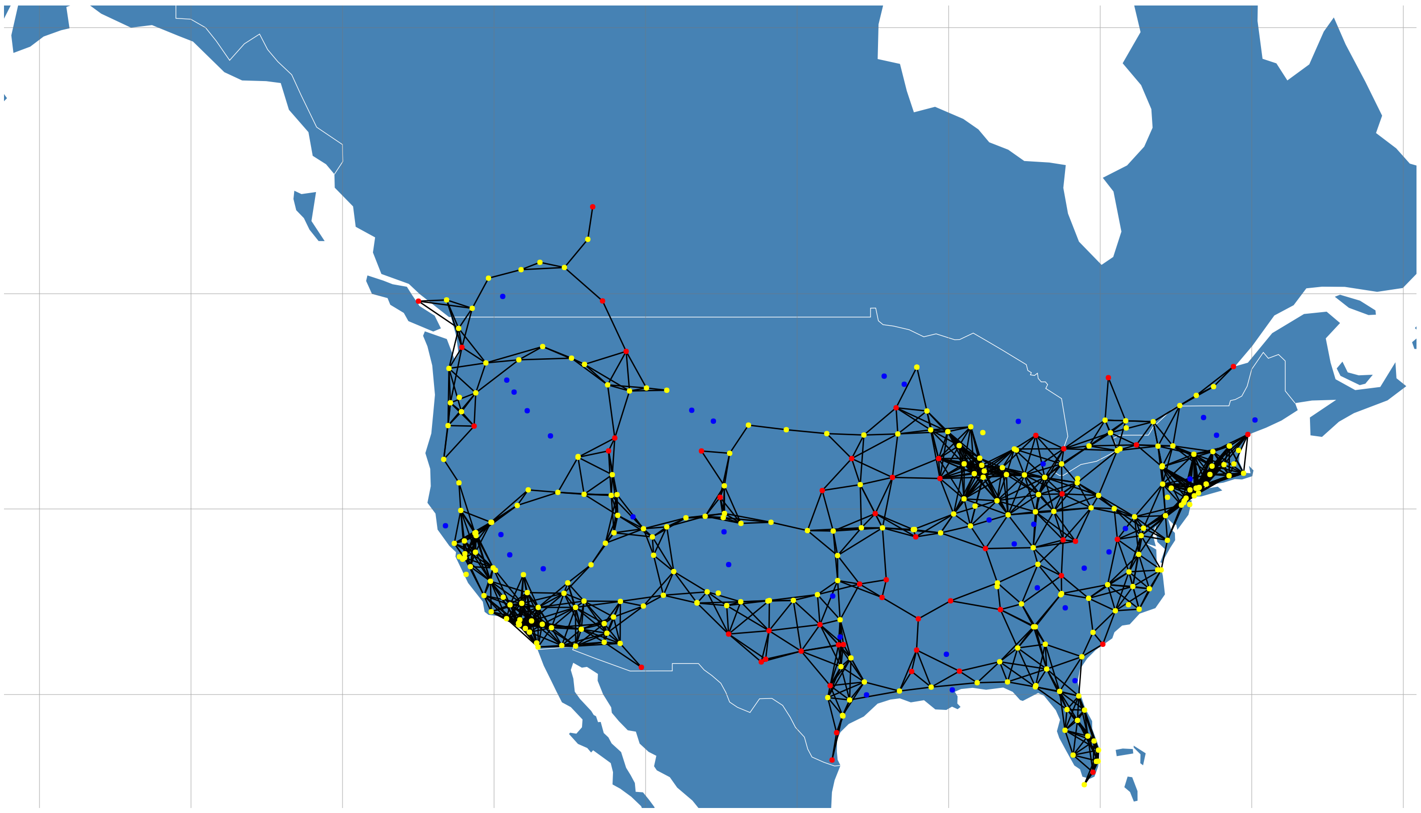
With the cost function attributes and similarity score defined, the problem becomes a maximization optimization - maximizing the similarity score by tuning the attribute weights of the cost function. However, our "function" can be thought of to be discrete in nature. Conceptually, we are selecting a single, discrete location to expand to at each growth step, and this defines the ultimate network and similarity score. Additionaly, this discrete function will not be very well-behaved, having a constant output for many sets of inputs, but becoming dramitacally different with a single small change in input. In my brief research review, I was unable to discover an algorithm suitable for such a mishaved optimization function, so we are left with attempting to brute force search for optimal parameter weights.

A weighting of 0.8 efficiency and 0.23 connectivity resulted in a similarity scores of 87%, when corrected for the network seed of 206 nodes results in correctly predicting 38/74 SCs, or 51%. Only being able to correctly model half of the expansion Superchargers isn't great, but considering the broad assumptions and limitations of the current model with the search space containing a maximum of 62% of real SCs, this optimization comes close to the theoretical maximum that the global best set of weights could model. Future work could consider refining these weights further.

The following graphic displays the modeled network overlaid with the actual SC network. Nodes in yellow are the SCs correctly predicted by the model, nodes in red are the SCs incorrectly predicted by the model and nodes in blue are the SCs in the real network not discovered by the model. It's interesting to note that a large fraction of the SCs not discovered by the model form connecting "bridges" across large stretches of the network. My intuition is these types of multiple node bridges require advanced planning of multiple future Supercharger expansions, something that is not currently considered in our model.

```
0.8714285714285689
0.5135135135135039

37.99999999999929
```



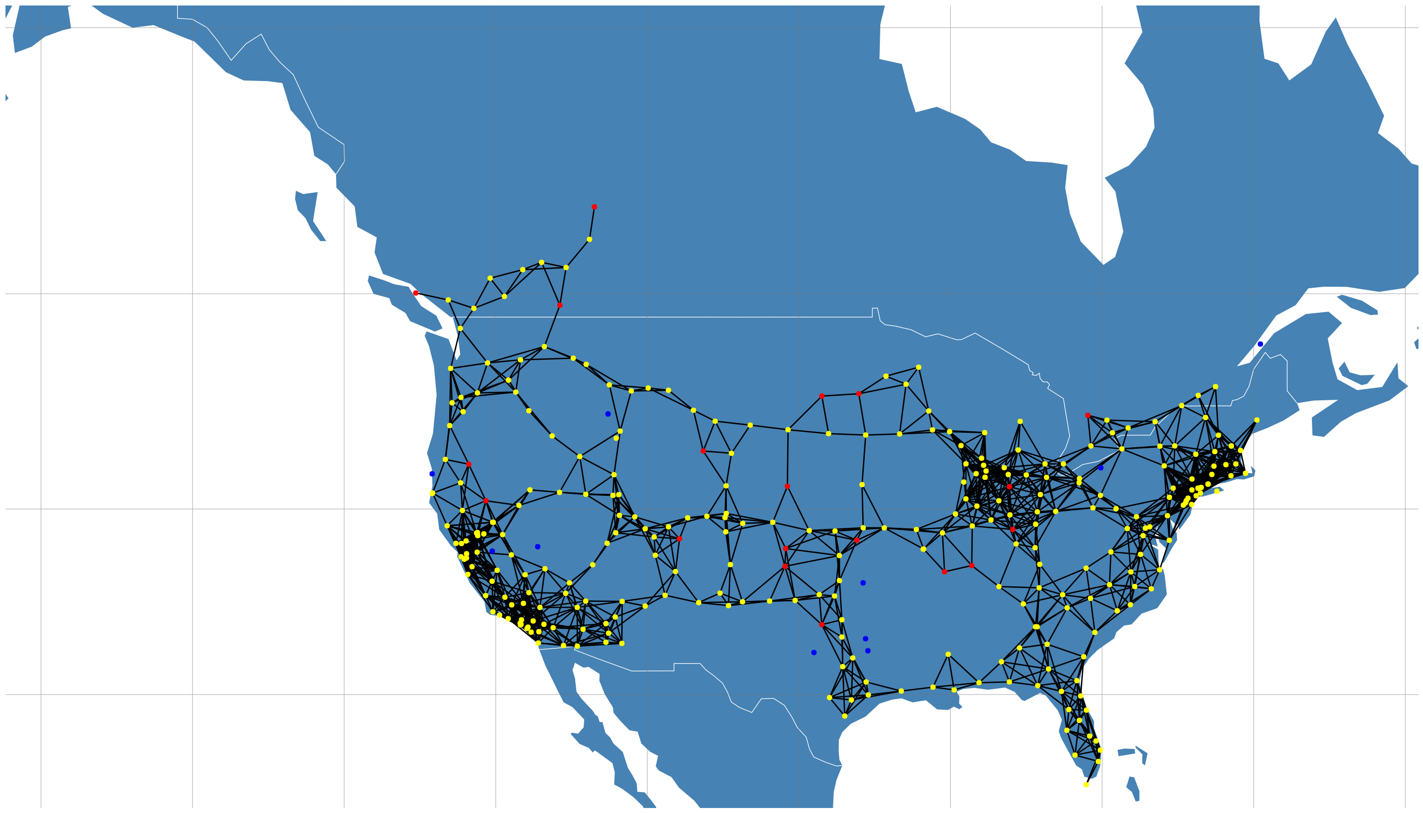
Final Model

We can validate the performance of our model by comparing the held out data of 22 Tesla Supercharger currently under construction, with the predicted network of the same number of SCs using our model. The graphic below displays the models ability to correctly predict 12 of 22 futures SCs, or 55%, which is a pretty reasonable performance considering the complexity of the decision process we are trying to model.

With this final prediction model, and using an estimated rate of 12 new Superchargers per month (ie 280 new SCs in 24 months - doubling the network size in 2 years) we can develop a tool that allows users to vizualize the predicted growth of the Supercharger network. This tool can be found here (http://cole-maclean.github.io/MAI-CN/).

```
0.96688741721854630
0.5454545454545894

12.000000000000966
```



Conclusions

Analysis Summary

In this analysis, we've explored use cases and applications of geohashing, how to build hueristics to reduce potential search spaces, and attempted to build models using both classification and optimization techniques. We've also discovered that modeling complex decisions with partial data is, well, hard. In the end, a model was developed able to reflect the connectivity and efficiency growth mechanisms of the Tesla network, but is unable to account for other underlying decision attributes that exist when Tesla decides where to build their next SC. Although the model does not perfectly predict an ideal network expansion location, it can be used as a foundation to further develop with other sources of information, and to identify locations that may not have been otherwise explored.

Tool Review

D3.js
The more I use the JavaScript visualization library D3.js, the more I am amazed by its ability to build vizualizations exactly how I envision them, without compromise. Anything that I sketch with pen and paper, I've been able to emulate with D3.js. Like many others, I've found the learning curve to be quite steep for this tool, but the final results and limitless flexibility makes my imagination the limiting factor when building data vizulaliations - not the tools I use to make them a reality.

pickle
As part of this analysis, there were many use cases for having the ability to store an object to disk for storage and later use, essentionally caching an object with all of its precomputed attributes and methods. The python libraray, pickle (https://docs.python.org/2/library/pickle.html) allows you to do just that. This library makes it possible to very easily dump and load python objects to and from disk, so that a python object or class that takes a large amount of computational resources to build can be built once, and subsequently used infinitely many times without the need to recompute the object. One common use case of this library is in conjuction with sklearn training algorithms, where we train a model like an SVM or logistic regression, store the pre-trained model to disk and load it when an application requires predicitons on other datasets. This avoids the need to retrain the model everytime we fire up a new application. In this analysis, I utilized this technique to store a pre-computed "seed" network that was used to build other networks from classification or optimization models having different parameters. This pre-computed seed stored the search space and network attributes for each potential growth location, making any subsequent network growth step trivial to compute. This was most important for the brute force optimization model, when 100's of different networks were iteratively generated and similarity scores compared.

multiprocessing
This has been the most computationally challenging problem I've yet worked on and forced me to think hard about code architecture and to review other tools for handling computationally difficult problems. One tool I discovered was python's multiprocessing (https://docs.python.org/2/library/multiprocessing.html) library. This tool makes it fairly simple to run multiple processes concurrently on each core of a machine. I had some success in utilizing this library to generate multiple networks simutaneously, but discovered that concurrent programs need to be designed as such from the beginning. Because I was writing to a file on disk at some stages in my program, so called "race conditions" existed in my program where multiple processes were attempting to read and write to the same file at the same time. This caused file corruptions, and made the multiprocessing pretty much unusable. I'll need to study more about parallel program design and architecture before taking full advantage of this python library.

google maps APIs
This analysis would not have been possible without google maps APIs. I used both the directions (https://developers.google.com/maps/documentation/directions/) and geoencoding (https://developers.google.com/maps/documentation/geocoding/intro) APIs to great success. Especially with the directions API, google has abtracted away very challenging geographical problems and provides abundant data in a very clean and useable form. Both APIs are free for a base amount of daily requests.

Lessons Learned

Cache All The Things
Due to the computationally challenging nature of complex networks, I was forced to design my programs such that they could finish running sometime in 2016. One lesson I learned from this is that recomputing the same thing over and over can quickly add up to alot of wasted time. Reviewing my codebase and caching everything that could be cached, like google map API calls, network seach spaces and attributes, and fully pre-compiled seed networks, I was able to significantly improve the speed of the programs used in this analysis.

I'll need to research caching principles and best practices to further utilize this concept.

Codebases Only Grow
I am very surprised by how large the codebase for this analysis has become. Initially, I envisioned a simple class with some data manipulation scripts built around it, all of which I could hold and manage in my head. This turned out not to be the case, and I spent alot of time debugging errors that crept into the programs because it became too large to keep track of each moving part. I would have benifited greatly if I designed the program with growth in mind from the beginning, utilizing testing and better commenting from the start. Testing is something I've had little practice with, and need to invest more time into learning and applying.

The Bias for Positive Results
I became somewhat disappointed through the course of this analysis, as each avenue of attempted models, many of which are not detailed here, led to unsatisfactory results. I had hoped to build a more powerfully predictive model with the current availible data, but was unable to find a combination of data points able to provide sufficient explanatory power for such a model. I found myself attempting to influence the analysis to provide better results then actually existed, by explaining away outliers that made my model worse, but keeping those that made it better, or by adjusting the size of the train/test data split to produce a better looking confusion matrix. Looking back, these actions seem almost to have occured unconsciencely, and it's quite possible I'm missing other manipulations that may have crept into this analysis. I was (and admittably still am), biased towards obtaining a "positive" result from the analysis. But the point of this analysis was not to build a perfectly predictive model - it was to explore what is possible build and learn from the data, and more importantly for me, to learn the techniques and lessons I've listed here. I'll have to make an effort to remember this lesson when the source of the bias is not only my own, but when external pressures exist influencing the biases of an analysis.

Future Work

Additional Modeling Techniques
Here we've explored 2 techniques for modeling this decision process, but there are other techniques that exist which may provide better results. Two that come to mind are MCDA and Complex Network modeling. In MCDA, there exists techniques of pairwise preference modeling, where given a set of pairs of alternatives with their attribute values and the prefered alternative labelled, it is possible to learn the preference model of a decision maker. This (http://link.springer.com/article/10.1007/s10994-013-5365-4) paper provides a very good overview of this technique and provides a comparison to the Machine Learning clasification technique we used in this analysis. Alternatively, techniques for modeling complex networks that take into account node attribute value exist, such as the Multiplicative Attribute Graph Model (https://cs.stanford.edu/people/jure/pubs/mag-im12.pdf). It would be interesting to compare the results of these techniques with the ones utilized here.

Multi-node Growth Modeling
The current model is only able to model the decision of adding a single Supercharger at each step in network growth, but this is not a realistic model of the actual decision process. Tesla will consider the effects of added multiple Superchargers in their plans, as looking two or three SC in the future can build a bridge in the network that increases the networks utility far more than adding 3 locally maximal seperate SCs. This type of future planning increases the search space of potential networks exponentially and becomes analogous to the next move search space computational problem tackled by google's AlphaGo team. Perhaps techniques developed there could be utilized in these type of decision processes.

Incorporate More Data
The data model and analysis here provide a foundation for developing future models that can incorporate more and better discriminating datasets to build a better predictive growth model of the Tesla Supercharger Network. For now, I hope others have fun exploring this analysis and attempting to predict when their city might be getting a SC.