Model Clustering via Group Lasso

David Hallac
hallac@stanford.edu
CS 229 Final Report

1. INTRODUCTION

As datasets get larger and more intricate, many classical methods of analysis begin to fail due to a lack of scalability and/or robustness. This in particular holds true with networks, which allow us to represent the complicated relationships between different pieces of information. The challenge lies in developing models sophisticated enough to uncover patterns, general enough to work well independent of the input, and capable of scaling to the immense networks that today's questions require.

While some network properties are easily and efficiently computed, most non-trivial features, even things as simple as k-means clustering, rely on heuristics which do not guarantee the optimal solutions. In other words, they are not convex. These approaches often yield solutions which are "good enough", but there is no way (in polynomial time) to know how suboptimal these answers are, i.e how much better we can do. Furthermore, as we delve deeper into network analysis, we hope to model and discover even more complex features. These will require new techniques and algorithms where fast, guaranteed convergence will be of the utmost importance.

In general, solving these large-scale problems is not easy. With the complicated coupling imposed by real world networks, it is not feasible to assume any specific structure. Therefore it is necessary to be able to solve this problem on any type of network, given a variety of possible constraints and conditions. To do so, we formulate it as a convex optimization problem. Unfortunately, optimization solvers do not scale well past several thousand unknown variables, too small for many real world applications. Therefore, standard methods will not work. We instead need to develop a distributed algorithm to optimize networks of this size.

There are several reasons why this is just now coming to the forefront of research interests. With increasing availability of data and cheaper computing costs, it is becoming more viable to perform complex large-scale analysis. However, as this field is still relatively new, it has been easy to use the prior heuristics and consider the local optima as sufficient solutions. Scaling up these more mathematically rigorous methods requires an integration of two fields, network analysis and convex optimization. Most of the related work has focused on specific instances and applications of optimization problems, rather than formulating more general solutions.

In this project, we develop a framework for solving optimization problems on networks. This particular approach is best used when there is prior knowledge that some sort of clustering exists, even if we know nothing else about it. We use a distributed algorithm which is guaranteed to converge to the global optimum, and a similar distributed non-convex one which has no guarantees but tends to perform very well. Then, we apply this method to two common machine learning problems, binary classification and predicting housing prices, and compare our results to common baselines.

2. CONVEX PROBLEM DEFINITION

Given a connected, undirected graph $\mathcal{G}$, consisting of m nodes and n edges with weights $w_{jk} \in \mathbf{R}_+$, we solve for a set of variables $x_i \in \mathbf{R}^p, i = 1, \dots, m$, one per node. These can represent parameters in a statistical model, optimal actions to undertake, etc. Each node has a closed proper convex objective function $f_i \colon \mathbf{R}^p \to \mathbf{R} \cup \{\infty\}$, which we want to minimize. Additionally, there is a proportional penalty whenever two connected nodes have different values of x. This can be written as the optimization problem

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk} \|x_j - x_k\|_2, \qquad (1)$$

where $\lambda \in \mathbf{R}_+$ is the regularization parameter, $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the edge set. This problem is convex in the variable $x = (x_1, \dots, x_m) \in \mathbf{R}^{mp}$. We let $x^\star$ denote an optimal solution.

Regularization Path. Although λ can be incorporated into the w_{ij}'s by scaling the edge weights, it is best viewed separately as a single parameter which is tuned to yield different global results. At $\lambda = 0$, $x_i^\star$, the solution at node i, is simply any minimizer of f_i. This can be computed locally at each node, since when $\lambda = 0$ the network has no effect. At the other extreme, there can exist a $\lambda_{\text{critical}}$ such that for any $\lambda \geq \lambda_{\text{critical}}$, we have $x_i^\star = x^{\text{cons}}$, *i.e.*, the solution at every node is equal. Thus problem (1) turns into

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x), \qquad (2)$$

which is solved by $x^{\text{cons}} \in \mathbf{R}^p$. We refer to (2) as the consensus problem and to x^{cons} as the consensus solution. It can be shown that, if (2) has a finite solution, $\lambda_{\text{critical}}$ exists and must be finite. For λ's in between $\lambda = 0$ and $\lambda_{\text{critical}}$, the family of solutions follows a trade-off curve and is known as the regularization path, though it is sometimes referred to as the clusterpath [3].

Group Lasso and Clustering. The ℓ_2-norm penalty over the edge difference, $\|x_j - x_k\|_2$, is called group lasso [11]. It incentivizes the differences between connected nodes to be exactly zero, rather than just close to zero, yet it does not penalize large outliers (in this case, node values being very different) too severely. An edge difference of zero means that $x_j = x_k$. When many edges are in consensus like this, we have clustered the nodes into groups with equal x values. The outliers then refer to edges between nodes in different clusters. The sizes of these clusters depend on λ. Average cluster size gets larger as λ increases, until at $\lambda_{\text{critical}}$ the consensus solution can be thought of as a single cluster for the entire network. Even though increasing λ is most often agglomerative, cluster fission may occur, so the clustering pattern is not strictly hierarchical [7].

This setup is similar to other regularization models, but leads to different results. The ℓ_2-norm considers the entire x_i as one entity, whereas an ℓ_1-norm, called the lasso [10], would have treated each element of x_i separately. This would yield element-wise instead of node-wise stratification. Had the ℓ_2-norm been squared, it would be Laplacian regularization. However, this would attempt to smooth out the results, which also would not lead to the clustering behavior that we are seeking.

Inference on New Nodes. After we have solved for x, we can interpolate the solution to predict x on new nodes, for example during cross-validation on a test set. Given a new node j, all we need is its location within the network; that is, the neighbors of j and the edge weights. With this information, we treat j like a dummy node, with $f_j(x_j) = 0$. We solve for x_j just like in problem (1) except without the objective function f_j, so the optimization problem becomes

$$\text{minimize} \quad \sum_{k \in N(j)} w_{jk} \|x - x_k\|_2. \tag{3}$$

This estimate of x_j can be thought of as a weighted median of j's neighbors' solutions. This is called the Weber problem, and it involves finding the point which minimizes the weighted sum of distances to a set of other points. It has no analytical solution when j has more than two neighbors, but it can be readily computed even for large problems.

3. PROPOSED SOLUTION

Since the focus of this project is on the machine learning applications (and due to space limitations), we will not delve into the derivations of the algorithms. Instead, we will give a high-level description of how it works, and provide the distributed algorithm to compute it. Note that a distributed solution is necessary so that computational and storage limits do not constrain the scope of potential applications. We use a well-established method called Alternating Direction Method of Multipliers (ADMM) [1] [9]. With ADMM, each individual component solves its own private objective function, passes this solution to its neighbors, and repeats the process until the entire network converges. There is no need for global coordination except for iteration synchronization, so this method can scale to arbitrarily large problems.

Algorithm 1 Regularization Path

```
initialize Solve for $x^\star$ at $\lambda = 0$.
set $\lambda = \lambda_{\text{initial}}$; $\alpha > 1$.
repeat
    Use ADMM to solve for $x^\star(\lambda)$, starting at previous solution.
    Stopping Criterion. quit if all edges are in consensus.
    Set $\lambda := \alpha\lambda$.
return $x^\star(\lambda)$ for $\lambda$ from 0 to $\tilde{\lambda}_{\text{critical}}$.
```

We solve this problem along the entire regularization path to gain insight into the network structure. For specific applications, this may also help decide the correct value of λ to use, for example by choosing the λ which minimizes the cross-validation error. We begin the regularization path at $\lambda = 0$ and solve for an increasing sequence of λ's. We know when we have reached $\lambda_{\text{critical}}$ because a single x^{cons} will be the optimal solution at every node, and increasing λ no longer affects the solution. This will sometimes lead to a stopping point slightly above $\lambda_{\text{critical}}$, which we can denote $\tilde{\lambda}_{\text{critical}}$. Solving for the exact $\lambda_{\text{critical}}$ is impractical for all but the smallest problems . It is okay if $\tilde{\lambda}_{\text{critical}} \geq \lambda_{\text{critical}}$, though, since they will both yield the same result, the consensus solution.

At each step in the regularization path, we solve a single convex problem, a specific instance of problem (1) with a given λ, by ADMM. The entire process is outlined in Algorithm 1. The ADMM solution can be treated as a black box which provides a scalable, distributed solution to this large convex problem, and is guaranteed to converge to the global optimum.

4. NON-CONVEX EXTENSION

In many applications, we are using the group lasso as an approximation of the ℓ_0-norm [2]. That is, we are looking for a sparse solution where relatively few edge differences are non-zero. However, once an edge does break off, we do not care about the magnitude of $\|x_i - x_j\|_2$. The lasso has a proportional penalty, which is the closest that a convex function can come to approximating the ℓ_0-norm. However, once we've found the true clusters, this will "pull" the different clusters towards each other through their mutual edges. If we replace the group lasso penalty with a monotonically nondecreasing concave function $\phi(u)$, where $\phi(0) = 0$ and whose domain is $u \geq 0$, we come even closer to the ℓ_0 . However, this new optimization problem,

$$\text{minimize} \quad \sum_{i \in \mathcal{V}} f_i(x_i) + \lambda \sum_{(j,k) \in \mathcal{E}} w_{jk} \phi\left(\|x_j - x_k\|_2\right), \tag{4}$$

is not convex. ADMM is therefore not guaranteed to converge, and even if it does, it need not be to the global optimum. It is in some sense a "riskier" approach. In fact, different initial conditions on x, u, z, and ρ can yield quite different solutions. However, as a heuristic, a slight modification to ADMM has been shown to perform well in these cases. We will use this approach in the specific case where $\phi(u) = \log(1 + \frac{u}{\epsilon})$, where ϵ is a constant scaling factor, and compare our results to the baselines and convex case.

5. IMPLEMENTATION

We test our approach on two different examples. To run these experiments, we built a module combining Snap.py [8] and CVXPY [5]. The network is stored as a Snap.py structure, and the ADMM updates are run in parallel using CVXPY.

6. EXPERIMENTS

We now examine this method to see two potential problems that this approach can solve. First, we look at a synthetic example in which we gather statistical power from the network to improve classification accuracy. Next, we see how it can apply to a geographic network, allowing us to gain insights on residential neighborhoods by looking at home sales.

6.1 Network-Enhanced Classification

Dataset. In this synthetic dataset, there are 1000 randomly generated nodes, each with its own classifier, a support vector machine (SVM) in $\mathbf{R}^{50}$ [4]. Given an input $w \in \mathbf{R}^{50}$, it tries to predict $y \in \{-1, 1\}$, where

$$y = \text{sgn}(a^T w + a_0 + v),$$

and $v \in \mathcal{N}(0, 1)$, the noise, is independent for each trial. An SVM involves solving a convex optimization problem from a set of training examples to obtain $x = \begin{bmatrix} a \\ a_0 \end{bmatrix} \in \mathbf{R}^{51}$. This defines a separating hyperplane to determine how to classify new inputs. There is no way to counter the noise v, but an accurate x can help us predict y from w reasonably accurately. Each node determines its own optimal classifier from a training set consisting of 25 (w, y)-pairs, which we use to solve for x at every node. To evaluate our results, there is a test set of 10 (w, y)-pairs per node. All elements in w, a, and v are drawn independently from a normal distribution with an identity covariance matrix, with the y values dependent on the other variables.

Network. The 1000 nodes are split into 20 equally-sized clusters, each with 50 nodes. Each cluster has a common underlying classifier, $\begin{bmatrix} a \\ a_0 \end{bmatrix}$, while different clusters have independent a's. If i and j are in the same cluster, they have an edge with probability 0.5, and if they are in different clusters, there is an edge with probability 0.01. Overall, the network has a total of 17079 edges, with 28.12% of the edges connecting nodes in different clusters.

Our goal. Each node has nowhere near enough information to generate a robust classifier, since there are twice as many dimensions as there are training examples. We hope that the nodes can, in essence, "borrow" training examples from their relevant neighbors to improve their own classifiers. Of course, neighbors in different clusters will provide misleading information. These are the edges which should break off, yielding different x's at the two connected nodes. Note that even though this is a synthetic example, the large number of misleading edges means this is far from an ideal case. This problem can be thought of as an example of generic classification where we have incomplete/incorrect information. Each node needs to solve its own optimization problem, simultaneously determining which information to include and which to ignore.

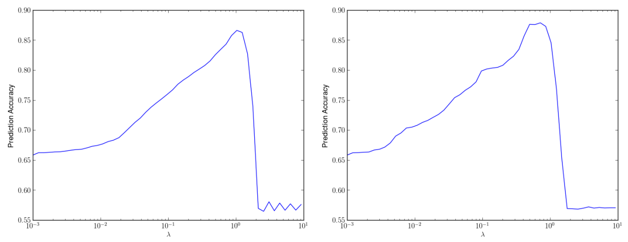
Optimization Parameter and Objective Function. The optimization parameter $x = \begin{bmatrix} x_a \\ x_0 \end{bmatrix} = \begin{bmatrix} a \\ a_0 \end{bmatrix}$ defines our estimate for the separating hyperplane for the SVM. Each node then solves its own optimization problem, using its 25 training examples. At node i, f_i is defined [6] as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|x_a\|_2^2 + \sum_{i=1}^{25} c\|\varepsilon_i\|_1 \\ \text{subject to} & y^{(i)}(a^T x_a + x_0) \geq 1 - \varepsilon_i, \quad i = 1, \ldots, 25, \end{array}$$

where c is the soft-margin threshold parameter empirically found to perform well on a common model.We solve for 51 + 25 = 76 variables at each node, so the total problem has 76,000 unknown variables.

Results. To evaluate performance, we find prediction accuracy on the test set of 10,000 examples (10 per node). We plot percentage of correct predictions vs. λ, where λ is displayed in log-scale, for the entire regularization path from $\lambda = 0$ to $\lambda \geq \lambda_{\text{critical}}$, as shown in Figure 1. In this example, these two extremes represent important baselines.

At $\lambda = 0$, each node only uses its own training examples, ignoring all the information provided by the neighbors. This is just a local SVM, with only 25 training examples to estimate a vector in $\mathbf{R}^{51}$. This leads to a prediction accuracy of 65.90% and is clearly a suboptimal approach. When $\lambda \geq \lambda_{\text{critical}}$, the problem leads to a common x at every node, which is equivalent to solving a global SVM over the entire network. Even though we have prior knowledge that there is some underlying clustering structure, this assumes the entire graph is coupled together and does not allow for any edges to break. This common hyperplane at every node yields an accuracy of 57.10%, which is barely an improvement over random guessing.



(a) Convex (b) Non-convex

Figure 1: SVM regularization path

Method	Maximum Prediction Accuracy
Local SVM ($\lambda = 0$)	65.90%
Global SVM ($\lambda \geq \lambda_{\text{critical}}$)	57.10%
Convex	86.68%
Non-convex	87.94%

Table 1: Prediction accuracy compared to baselines

In contrast, both the convex and non-convex cases perform much better for λ's in the middle. From Figure 1, we see a distinct shape in both the convex and nonconvex regularization paths. As λ increases, the accuracy steadily improves, until a peak of around $\lambda = 1$. Intuitively, this represents the point where the algorithm has approximately split the nodes into their correct clusters, each with its own classifier. As λ goes past one, there is a rapid drop off in prediction accuracy, due to the different clusters "pulling" each other together. The maximum prediction accuracies in this plots are 86.68% (convex) and 87.94% (nonconvex). These results are summarized in Table 1.

6.2 Spatial Clustering with Regressors

Dataset. We look at a list of real estate transactions over a one-week period in May 2008 in the Greater Sacramento area[1]. This dataset contains information on 985 sales, including latitude, longitude, number of bedrooms, number of bathrooms, square feet, and sales price. However, as often happens with real data, we are missing some of the values. 17% of the home sales are missing at least one of the regressors. To verify our results, we remove a random subset of 200 houses as a test set.

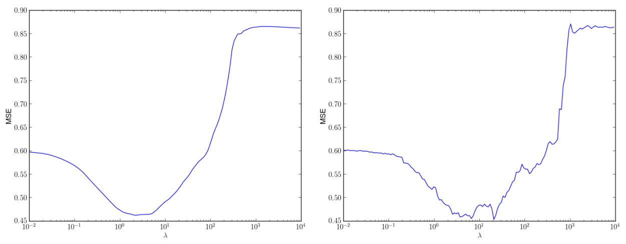
Network. We build the graph by using the latitude/longitude coordinates of each house. After removing the test set, we connect every remaining house to the five nearest homes with an edge weight of

$$w_{ij} = \frac{1}{d_{ij} + \epsilon},$$

where d_{ij} is the distance in miles from house i to house j and $\epsilon = 0.1$ bounds the weights between 0 and 10. If house j is in the nearest neighbors of i, there is an undirected edge regardless of whether or not house i is in the set of nearest neighbors of j, which may or may not be the case. The resulting graph leaves 785 nodes and 2447 edges, and it has a diameter of 61.

Our goal. We attempt to estimate the price of the house based on the spatial network and the set of regressors (number of bedrooms, bathrooms, and square feet). The reason that the spatial

[1]Data comes from `http://support.spatialkey.com/spatialkey-sample-csv-data/`



(a) Convex (b) Non-convex

Figure 2: Regularization path for housing data

network will help is that house prices often cluster together along neighborhood lines. For reasons relating to taxes, security, school districts, etc, similar houses in different locations can have drastically different prices. In this example, the clustering occurs when nearby houses have similar pricing models, while edges that break will be between those in different neighborhoods. Simultaneously, as houses group together, each cluster will build its own local linear regression model to predict the price of new houses in the area. This way, by using our method to combine a spatial network with the housing regressors, we can obtain a better price estimate than other common methods.

Optimization Parameter and Objective Function. At each node, we solve for a variable

$$x = \begin{bmatrix} a & b & c & d \end{bmatrix}^T,$$

which shows how the price of the house depends on the set of regressors. For simplicity, the price and all regressors are standardized to zero mean and unit variance. Any missing features are ignored by setting the value to 0, the average. The price estimate is given by

$$\overline{\text{price}} = a \times \text{Bedrooms} + b \times \text{Bathrooms} + c \times \text{SQFT} + d,$$

where the constant offset d is the "baseline", the normalized price for which a house in that location would sell for with a global average number of bedrooms, bathrooms, and square feet. We aim to minimize the mean square error (MSE) of our price estimate. To prevent overfitting, we regularize the a, b, and c terms, everything besides the offset. The objective function at each node then becomes

$$f_i = \|\overline{\text{price}} - \text{price}\|_2^2 + \mu \left\| \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x \right\|_2^2$$

where μ is a regularization parameter found to perform well on a common model.

To predict the prices for the test set houses, first connect each new house to the 5 nearest sales, weighted by inverse distance, just like before. We then infer the value of x by solving problem (3) and use it to estimate the sales price.

Results. We plot the MSE of our test set price estimates vs. λ in Figure 2 for the convex and nonconvex formulations of the problem. Once again, the two extremes of the regularization path are relevant baselines.

At $\lambda = 0$, the regularization term in $f_i(x)$ insures that the only non-zero element of x is d. Therefore, the weights of the regressors on the other features are 0, and d_i equals the price of house i. Our estimate for each new house is simply the weighted median price of the 5 nearest homes, which leads to an MSE of 0.60127 on the test set. This ignores the regressors and is a prediction based solely on spatial data. For large λ's, we are fitting a common linear model for all the houses. This is just regularized linear regression on the entire dataset and is the canonical method of estimating housing prices from a series of features. Note that this approach completely ignores the geographic network. As expected, in part due to the missing regressor information, it performs rather poorly, with an MSE of 0.86107. Note that since the prices are standardized with unit variance, a naive guess (with no information about the house) would be the global average. This would lead to an MSE of 1.0, so linear regression yields only a 13.9% improvement over this approach. The convex and non-convex methods are both maximized around $\lambda = 5$, with minimum MSE's of 0.46299 and 0.45393, respectively. The convex MSE is 46.2% lower than regularized linear regression and 23.0% lower than the network-only method. The non-convex MSE drops by 47.3% and 24.5%, respectively. The results are summarized in Table 2.

Method	MSE
Geographic ($\lambda = 0$)	0.60127
Regularized Linear Regression ($\lambda \geq \lambda_{\text{critical}}$)	0.86107
Naive Prediction (Global Mean)	1.0
Convex	0.46299
Non-convex	0.45393

Table 2: Mean squared error for housing price predictions

We can visualize the clustering pattern by overlaying the network on a map of Sacramento. We plot each sale with a marker, colored according to its corresponding x_i (converting each vector from $\mathbf{R}^4$ into a 3-dimensional RGB color, so houses with the same color are in consensus). With this, we see how the clustering pattern emerges. In Figure 3, we look at this plot for five values of λ between 0.1 and 1,000. In 3(a), λ is very small, so the neighborhoods have not yet formed. On the other hand, in 3(e) the clustering is very clear, but it doesn't perform well since it forces together neighborhoods which are very different.

Aside from outperforming the baselines, this method is also better able to handle anomalies, which certainly exist in this housing dataset. In particular, a subset of over 40 houses, all on one street in Lincoln, California, were "sold" on the same day for \$4,987 each. This is the type of problematic information that typically needs to be removed via a data cleansing process. Otherwise, it will have a large adverse effect on the results. Our approach is robust to these sorts of anomalies, and is in fact even able to detect them. In Figure 3(c), these outliers are confined to the cluster of houses in the northwest section of the plot, colored in black. Its effect on the rest of the network is limited. There is a very large edge difference between the anomalous cluster and its neighbors. However, when λ approaches $\lambda_{\text{critical}}$, these outliers are forced into consensus with the rest of the network, so we run into the same problem that linear regression did. Near the optimal λ, though, our method accurately classifies these anomalies, separates them from the rest of the graph, and builds separate and relatively accurate models for both subsets.

7. CONCLUSION AND FUTURE WORK

These results show that within one single framework, we can better understand and improve on several common machine learning/network analysis problems. The magnitude of the improvements over the baselines show that this approach is worth exploring further, as there are many potential ideas to build on. Within the ADMM algorithm, there are plenty of opportunities to improve speed, performance, and robustness. This includes finding closed-form solutions for common objective functions, automatically determining the optimal ADMM parameters, and even allowing edge objective functions $f_e(x_i, x_j)$ beyond just the weighted group lasso. From a machine learning standpoint, there are other ML problems that can be put into this framework, for example anomaly detection in networks, which was briefly touched upon in Section 6.2. Eventually, we hope to develop an easy-to-use interface that lets programmers solve these types of large-scale problems in a distributed setting without having to specify the ADMM details, which would greatly improve the practical benefit of this work.

Acknowledgements

This work is in collaboration with Stephen Boyd and Jure Leskovec.

8. REFERENCES

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine learning*, 3:1–122, 2011.

[2] E. Candes, M. Wakin, and S. Boyd. Enhancing sparsity by reweighed ℓ_1 minimization. *Journal of Fourier analysis and applications*, 14:877–905, 2008.

[3] E. Chi and K. Lange. Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, 2013.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[5] S. Diamond, E. Chu, and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization, version 0.2. `http://cvxpy.org/`, May 2014.

[6] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. pages 1391–1415, 2004.

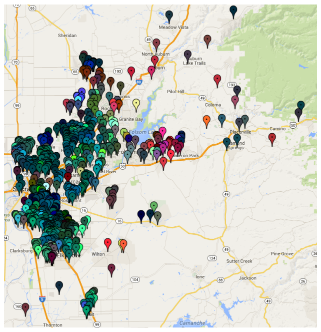
[7] T. Hocking, A. Joulin, F. Bach, and J. Vert. Clusterpath: an algorithm for clustering using convex fusion penalties. *28th International Conference on Machine Learning*, 2011.

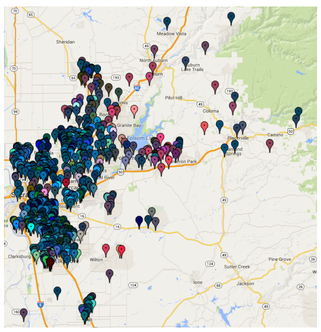
[8] J. Leskovec and R. Sosič. Snap.py: SNAP for Python, a general purpose network analysis and graph mining tool in Python. `http://snap.stanford.edu/snappy`, June 2014.

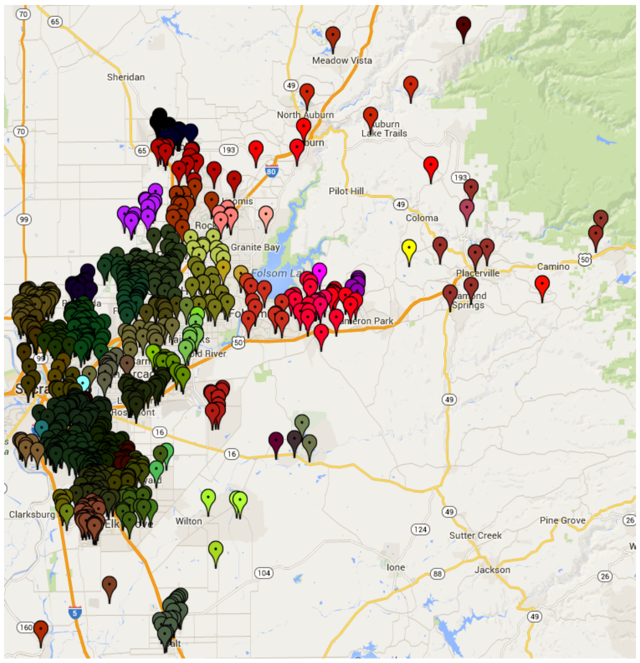
[9] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1:123–231, 2014.

[10] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, pages 267–288, 1996.

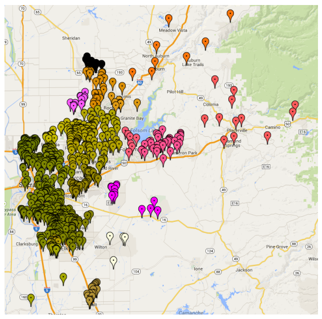
[11] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B*, 68:49–67, 2006.


(a) $\lambda = 0.1$


(b) $\lambda = 1$



(c) $\lambda = 10$


(d) $\lambda = 100$

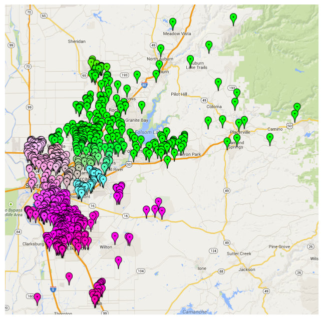

(e) $\lambda = 1000$

Figure 3: Regularization path clustering pattern