ECE 236A: Linear Programming

Prototype Selection for Nearest Neighbors Classification

Shakthi Visagan
804 622 954

Prof.: C. Fragouli
T.A.: D. Basu

2018 December 7

1 Introduction

Our goal is to reimplement the algorithm *prototype selection for interpretable classification* developed by Bien and Tibshirani in [1]. We are motivated by the fact that k-nearest neighbors machine learning techniques involve calculating the neighborhood of a test sample relative to the instances of a data set using a distance metric (which is computationally intractable when the data set is large), and such large data sets may be uninterpretable for domain specialists performing a manual analysis. We instead identify a set of prototype sets, subsets of the labeled data set, that contains fewer instances but meaningfully represents the data set and the constitutive classes. That is to say, this is not a form of dimensionality reduction because we are not reducing the size of the data set by reducing the number of *variables* that describe the data; instead we reduce the number of *instances* or *samples*. Bien et al. suggest that the proposed method seeks to "capture the full variability of a class while avoiding confusion with other classes" [1]. It is also of interest to study integer programs for optimization, which are essentially linear programs (the focus of this class) with the additional constraint that the variables are integral.

2 Problem Statement

We are given a training set of N points or samples, $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, where each instance has p features or variables describing it, $\boldsymbol{x}_n \in \mathbb{R}^p$. Each instance in the training set has a corresponding class label, $y_1, y_2, \ldots, y_N$, where we have L labels to choose from, $y_n \in \{1, 2, \ldots, L\}$. We want to return prototype sets $\mathcal{P}_l \subseteq \mathcal{X}$ sets for each class $l \in \{1, 2, \ldots, L\}$. Ultimately, with the resulting set of prototype sets $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_L\} \subseteq \mathcal{X}$, one should have a more interpretable, compressed or condensed form of the original unmanageable data set.

With the new set $\boldsymbol{\mathcal{P}}$, we can again perform k-nearest neighbors analysis and classify a test point, $\hat{\boldsymbol{x}} \in \mathbb{R}^p$, that is, obtain the label, $\hat{y}$, by finding the nearest *prototype* instead of the nearest *sample* and assigning it the corresponding class:

$$\hat{y} = \underset{l \in \{1,2,\ldots,L\}}{\arg\min} \left\{ \min_{\boldsymbol{z} \in \mathcal{P}_l} \{\text{dist}(\hat{\boldsymbol{x}}, \boldsymbol{z})\} \right\}. \tag{1}$$

3 Problem Formulation

3.1 Set Cover Optimization

Bien et al. includes an interpretation of prototype selection from the set cover perspective [1]. Here we study a set (sometimes referred to as the *universe*), $\mathcal{X}$, where the size of $\mathcal{X}$ is N, $|\mathcal{X}| = N$, and a collection of sets, S, whose union is equal to $\mathcal{X}$ and where the size of S is M, $|S| = M$. We say that S forms a *cover* of $\mathcal{X}$, and our goal is to find the smallest subcover of $\mathcal{X}$. What we do is identify a mapping, $B(\boldsymbol{x}_n)$, that take elements of $\mathcal{X}$ to elements of S. Ultimately our goal is to find a subset, $\mathcal{X}_J \subseteq \mathcal{X}$, that satisfies the following:

$$S = \bigcup_j B(\boldsymbol{x}_j), \quad \forall \boldsymbol{x}_j \in \mathcal{X}_J \subseteq \mathcal{X}. \tag{2}$$

That is to say, our goal is to find a subset, $\mathcal{X}_J$, of our universe, $\mathcal{X}$, such that the union of the mappings of the elements in $\mathcal{X}_J$ should be S, the cover of our universe. Since we are trying to find the *smallest* subcover of $\mathcal{X}$, we write it as the optimal of a integer program. We introduce indicator variables α_n where $\alpha_n = 1$ if $\boldsymbol{x}_n \in \mathcal{X}_J$ and $\alpha_n = 0$ otherwise (picking $\boldsymbol{x}_n \in \mathcal{X}$ to be in our subcover $\mathcal{X}_J$ or not). If our goal is to find the smallest subcover $\mathcal{X}_J$, then we

minimize $\sum_{n=1}^{N} \alpha_n$. We also need to make sure that every element, $\boldsymbol{x}_n$, in our universe, $\mathcal{X}$, is covered by the mappings of the elements in our optimal subcover at least once. Using our indicator variable, we write the following integer program:

$$
\begin{array}{lllll|l}
\text{minimize} & \displaystyle\sum_{n=1}^{N} & \alpha_n & & & \Rightarrow \text{find the smallest subcover of } \mathcal{X}, \\
\text{subject to} & \displaystyle\sum_{j\,:\,\boldsymbol{x}_n \in B(\boldsymbol{x}_j)} & \alpha_j & \geq 1, & \forall \boldsymbol{x}_n \in \mathcal{X}, & \Rightarrow \text{each } \boldsymbol{x}_n \text{ in our universe, } \mathcal{X}, \text{ should be covered by a mapping, } B(\boldsymbol{x}_j), \text{ at least once, with } \boldsymbol{x}_j \in \mathcal{X}_J, \textit{ or} \text{ for each } \boldsymbol{x}_n \in \mathcal{X}, \text{ count the number of mappings } B(\boldsymbol{x}_j) \text{ where } \boldsymbol{x}_n \in B(\boldsymbol{x}_j) \text{ and ensure that it is included in at least one mapping,} \\
 & & \alpha_n & \in \{0,1\}, & \forall \boldsymbol{x}_n \in \mathcal{X}, & \Rightarrow \alpha_n = 1 \text{ if } \boldsymbol{x}_n \in \mathcal{X}_J \text{ and } \alpha_n = 0 \text{ otherwise.}
\end{array}
\tag{3}
$$

3.2 Prototype Selection from Set Cover Formulation

To go from our set cover optimization introduction to the problem, we first see that the universe, $\mathcal{X}$, is simply our data set with N instances, $|\mathcal{X}| = N$. Furthermore, we also see that the cover we are mapping to is also our data set, $\mathcal{X}$. It is then easy to see that our desired subcover, $\mathcal{X}_J$, is our set of prototype sets, $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_L\} \subseteq \mathcal{X}$.

Here we also need to define the mapping, $B(\boldsymbol{x}_n)$, that now takes elements from our universe, $\mathcal{X}$, to its cover, $\mathcal{X}$. One possible mapping would be just an identity mapping where $B(\boldsymbol{x}_n) = \boldsymbol{x}_n \,:\, \boldsymbol{x}_n \in \mathcal{X}$, but it is obvious that doing so would not identify a compressed or "prototypical" version of our data set, $\mathcal{X}$. Good prototypes should represent the data *near* it, and thus it makes sense to write the mapping as a neighbourhood defining function. Without loss of generality, we can let the mapping be the following: $B(\boldsymbol{x}_n) = \{\boldsymbol{x}' \in \mathbb{R}^p \,:\, \text{dist}(\boldsymbol{x}_n, \boldsymbol{x}') \leq \varepsilon\}$. Now we map elements of our universe, $\boldsymbol{x}_n \in \mathcal{X}$, to balls of radius $\varepsilon > 0$ centered at $\boldsymbol{x}_n$. We can consolidate the above changes as the following, similar to (2):

$$
\mathcal{X} = \bigcup_j B(\boldsymbol{x}_j), \quad \forall \boldsymbol{x}_j \in \boldsymbol{\mathcal{P}} \subseteq \mathcal{X}. \tag{4}
$$

We can also rewrite the integer program from (3) as the following:

$$
\begin{array}{lllll|l}
\text{minimize} & \displaystyle\sum_{n=1}^{N} & \alpha_n & & & \Rightarrow \text{find the smallest set of prototype sets, } \boldsymbol{\mathcal{P}}, \text{ of the data set, } \mathcal{X}, \\
\text{subject to} & \displaystyle\sum_{j\,:\,\boldsymbol{x}_n \in B(\boldsymbol{x}_j)} & \alpha_j & \geq 1, & \forall \boldsymbol{x}_n \in \mathcal{X}, & \Rightarrow \text{each } \boldsymbol{x}_n \text{ in our data set, } \mathcal{X}, \text{ should be in a ball, } B(\boldsymbol{x}_j), \text{ at least once, with } \boldsymbol{x}_j \in \boldsymbol{\mathcal{P}}, \textit{ or} \text{ for each } \boldsymbol{x}_n \in \mathcal{X}, \text{ count the number of balls } B(\boldsymbol{x}_j) \text{ where } \boldsymbol{x}_n \in B(\boldsymbol{x}_j) \text{ and ensure that it is included in at least one ball,} \\
 & & \alpha_n & \in \{0,1\}, & \forall \boldsymbol{x}_n \in \mathcal{X}, & \Rightarrow \alpha_n = 1 \text{ if } \boldsymbol{x}_n \in \boldsymbol{\mathcal{P}} \text{ and } \alpha_n = 0 \text{ otherwise.}
\end{array}
\tag{5}
$$

With this linear program, we satisfy one obvious requirement of prototype selection, that our prototype set, $\boldsymbol{\mathcal{P}}$, covers our data set, $\mathcal{X}$, but also "compresses" the original data set. It also should minimize the size of the set as best as possible, according to the given ball radius, ε. However, what (5) fails to do is incorporate information about the classes. We augment the indicator variable and let $\alpha_n^{(l)} \in \{0,1\}$ where $\alpha_n^{(l)} = 1$ if $\boldsymbol{x}_n \in \mathcal{P}_l$ and $\alpha_n^{(l)} = 0$ otherwise, with $\mathcal{P}_l \in \boldsymbol{\mathcal{P}}$; that is, we pick $\boldsymbol{x}_n$ to be a prototype for class l or not. We again adapt our set cover rule from (2) and (4) through the following:

$$
\mathcal{X} = \bigcup_j \bigcup_l B(\boldsymbol{x}_j^{(l)}), \quad \forall \boldsymbol{x}_j^{(l)} \in \mathcal{P}_l \in \boldsymbol{\mathcal{P}} \subseteq \mathcal{X}. \tag{6}
$$

We augment our linear program to include the increase in the number of variables (from N to $N \times L$) to accommodate now for class selection, through the following:

$$
\begin{array}{lll|l}
\text{minimize} & \displaystyle\sum_{n=1}^{N}\sum_{l=1}^{L} \alpha_n^{(l)} & & \Rightarrow \text{find the smallest set of prototype sets, } \boldsymbol{\mathcal{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_L\}, \text{ of the data set, } \mathcal{X}, \\
\text{subject to} & \displaystyle\sum_{\substack{j\,:\,\boldsymbol{x}_n \in B(\boldsymbol{x}_j) \\ l = y_n}} \alpha_j^{(l)} \geq 1, & \forall \boldsymbol{x}_n \in \mathcal{X}, & \Rightarrow \text{each } \boldsymbol{x}_n^{(y_n)} \text{ in our data set, } \mathcal{X}, \text{ should be in a ball of the same class, } B(\boldsymbol{x}_j^{(l)}) : l = y_n, \text{ at least once, with } \boldsymbol{x}_j^{(l)} \in \mathcal{P}_l, \text{ a chosen prototype for class } l, \textit{ or } \text{for each } \boldsymbol{x}_n^{(y_n)} \in \mathcal{X}, \text{ count the number of balls of the same class } B(\boldsymbol{x}_j^{(l)}) : l = y_n \text{ where } \boldsymbol{x}_n^{(y_n)} \in B(\boldsymbol{x}_j^{(l)}) \text{ and ensure that it is included in at least one ball of the same class,} \\
 & \alpha_n^{(l)} \in \{0, 1\}, & \forall \boldsymbol{x}_n \in \mathcal{X}, l \in L. & \Rightarrow \alpha_n^{(l)} = 1 \text{ if } \boldsymbol{x}_n^{(l)} \in \mathcal{P}_l \text{ and } \alpha_n^{(l)} = 0 \text{ otherwise.}
\end{array}
\tag{7}
$$

3.3 Final Integer Program

The above integer program satisfies most of our requirements, the remaining one being that each neighborhood or ball should include as few training points of other classes. This can be rewritten as another constraint, similar to the method used in the first constraint where we show that each training point $\boldsymbol{x}_i^{(y_i)}$ is in a ball of its own class, $l = y_i$. Bien et al. take the time to list specifically the three requirements of their integer program, and parallel those requirements to the constraints and the objective in the program. We can also finalize the integer program with slack variables for feasible solving.

Prototype Selection for Interpretable Classification

For a given choice of $\mathcal{P}_l \in \boldsymbol{\mathcal{P}} \subseteq \mathcal{X}$, we consider the neighborhoods parameterized by ε at each $\boldsymbol{x}_n^{(l)} \in \mathcal{P}_l$. A desirable prototype set for class l is one that induces a set of neighborhoods that:

(a) covers as many training points of class l as possible,

$\Rightarrow$ each $\boldsymbol{x}_n^{(y_n)}$ in our data set, $\mathcal{X}$, should be in a ball of the same class, $B(\boldsymbol{x}_j^{(l)}) : l = y_n$, at least once, with $\boldsymbol{x}_j^{(l)} \in \mathcal{P}_l$, a chosen prototype for class l, or,

$\rightarrow$ for each $\boldsymbol{x}_n^{(y_n)} \in \mathcal{X}$, count the number of balls of the same class $B(\boldsymbol{x}_j^{(l)}) : l = y_n$, where $\boldsymbol{x}_n^{(y_n)} \in B(\boldsymbol{x}_j^{(l)})$ and ensure that it is included in at least one ball of the same class,

(b) covers as few training points as possible of classes other than l,

$\Rightarrow$ each $\boldsymbol{x}_n^{(y_n)}$ in our data set, $\mathcal{X}$, should be in as few balls of other classes, $B(\boldsymbol{x}_j^{(l)}) : l \neq y_n$, with $\boldsymbol{x}_j^{(l)} \in \mathcal{P}_l$, a chosen prototype for class l, or,

$\rightarrow$ for each $\boldsymbol{x}_n^{(y_n)} \in \mathcal{X}$, count the number of balls of different classes $B(\boldsymbol{x}_j^{(l)}) : l \neq y_n$, where $\boldsymbol{x}_n^{(y_n)} \in B(\boldsymbol{x}_j^{(l)})$ and ensure that it is included in as few balls of different classes,

(c) is sparse (i.e., uses as few prototypes as possible for the given ε)

$$
\begin{aligned}
& & & \text{(c)} & \text{(a)} & \text{(b)} & & \\
\min_{\alpha_n^{(l)}, \xi_n, \eta_n} \quad & \frac{1}{N} \sum_{n=1}^{N} \sum_{l=1}^{L} \alpha_n^{(l)} & & + \sum_{n=1}^{N} \xi_n & + \sum_{n=1}^{N} \eta_n & & \\
\text{subject to} \quad & \sum_{\substack{j \,:\, \boldsymbol{x}_n \in B(\boldsymbol{x}_j) \\ l = y_n}} \alpha_j^{(l)} & \geq 1 - \xi_n, & & & \forall \boldsymbol{x}_n \in \mathcal{X}, & \text{(8a)} \\
& \sum_{\substack{j \,:\, \boldsymbol{x}_n \in B(\boldsymbol{x}_j) \\ l \neq y_n}} \alpha_j^{(l)} & & \leq 0 + \eta_n, & & \forall \boldsymbol{x}_n \in \mathcal{X}, & \text{(8b)} \\
& \alpha_n^{(l)} \in \{0, 1\}, & \xi_n \geq 0, & & \eta_n \geq 0, & \forall \boldsymbol{x}_n \in \mathcal{X}, l \in L. &
\end{aligned}
\tag{8}
$$

We see that the second constraint (8b) is always tight (met with equality), and as such can be removed along with modifying the objective.

$$
\sum_{n=1}^{N} \eta_n = \sum_{n=1}^{N} \sum_{\substack{j \,:\, \boldsymbol{x}_n \in B(\boldsymbol{x}_j) \\ l \neq y_n}} \alpha_j^{(l)} = \sum_{j=1}^{N} \sum_{l=1}^{L} \alpha_n^{(l)} \sum_{n=1}^{N} 1 \left\{ \boldsymbol{x}_n \in B(\boldsymbol{x}_j) \cap \boldsymbol{x}_n \notin \mathcal{X}_l \right\} = \sum_{n=1}^{N} \sum_{l=1}^{L} \alpha_n^{(l)} \times \mid B(\boldsymbol{x}_n) \cap (\mathcal{X} \setminus \mathcal{X}_l) \mid \tag{9}
$$

Using the above, and a cost $C^{(l)}(j) = \frac{1}{N} + \mid B(\boldsymbol{x}_j) \cap (\mathcal{X} \setminus \mathcal{X}_l) \mid$, that counts how many times a point $\boldsymbol{x}_j^{(l)}$, if used as a prototype for $\mathcal{P}_l$, includes in its neighborhood points of other classes, we can rewrite the integer program for each class l:

$$
\begin{aligned}
\min_{\alpha_j^{(l)}, \xi_n} \quad & \sum_{j \,:\, \boldsymbol{x}_j \in \mathcal{X}_l} C^{(l)}(j) \alpha_j^{(l)} & + \sum_{\boldsymbol{x}_n \in \mathcal{X}_l} \xi_n & \\
\text{subject to} \quad & \sum_{j \,:\, \boldsymbol{x}_n \in B(\boldsymbol{x}_j)} \alpha_j^{(l)} & \geq 1 - \xi_n, & \\
& \alpha_j^{(l)} \in \{0, 1\}, & \forall \boldsymbol{x}_j \in \mathcal{X}_l, & \\
& \xi_n \geq 0, & \forall \boldsymbol{x}_n \in \mathcal{X}_l. &
\end{aligned}
\tag{10}
$$

4 Implementation

4.1 Randomized Rounding

In practice we implement the above, by relaxing the integral constraints $\alpha_j^{(l)} \in \{0,1\}$ to $\alpha_j^{(l)} \in [0,1]$, giving us a linear program. To recover the integer values we need for our indicator variable, we choose to round our optimal variables of the linear program.

At the end of our linear program, for each label $l \in L$, we have a solution of optimal set of $\alpha_j^{(l)}$ variables denoted as $\{\alpha_j^{(l)\star}\}$, an optimal set of ξ_n variables denoted as $\{\xi_n^\star\}$, and an optimal value denoted as $\text{OPT}_{\text{LP}}^{(l)}$. We can use those fractional probabilities found in the linear program to perform systematic rounding.

Randomized Rounding

For each class l, after solving the respective linear program from before, carry out the following algorithm for *randomized rounding*:

1. **INSTANTIATE** $A_1^{(l)} = \cdots = A_j^{(l)} = \cdots = A_m^{(l)} = 0$ and $S_1 = \cdots = S_n = \cdots = S_m = 0$
2. **FOR** $t = 1, 2, \ldots, 2 \times \log|\mathcal{X}_l|$:
   (a) **FOR** $i = 1, 2, \ldots, m$:
      i. ${}^{\text{temp}}A_i^{(l)} \sim \text{Bernoulli}(\alpha_i^{(l)\star})$
      ii. $A_i^{(l)} := \max\left({}^{\text{temp}}A_i^{(l)}, A_i^{(l)}\right)$
      iii. ${}^{\text{temp}}S_i \sim \text{Bernoulli}(\xi_i^\star)$
      iv. $S_i := \max\left({}^{\text{temp}}S_i, S_i\right)$
3. **CHECK** $\{A_j^{(l)}\}$ and $\{S_n\}$ to be feasible and $\text{OPT}_{\text{ROUND}}^{(l)} \leq 2 \times \log|\mathcal{X}_l| \times \text{OPT}_{\text{LP}}^{(l)}$:
   (a) $\displaystyle\sum_{j\,:\,\boldsymbol{x}_n \in B(\boldsymbol{x}_j)} A_j^{(l)} \geq 1 - S_n, \quad \forall \boldsymbol{x}_n \in \mathcal{X}_l$
   (b) $A_j^{(l)} \in \{0,1\}, \quad \forall \boldsymbol{x}_j \in \mathcal{X}_l$
   (c) $S_n \geq 0, \quad \forall \boldsymbol{x}_n \in \mathcal{X}_l$
   (d) $\displaystyle\text{OPT}_{\text{ROUND}}^{(l)} = \left(\sum_{j\,:\,\boldsymbol{x}_j \in \mathcal{X}_l} C^{(l)}(j) A_j^{(l)} + \sum_{\boldsymbol{x}_n \in \mathcal{X}_l} S_n\right) \leq 2 \times \log|\mathcal{X}_l| \times \left(\sum_{j\,:\,\boldsymbol{x}_j \in \mathcal{X}_l} C^{(l)}(j) \alpha_j^{(l)\star} + \sum_{\boldsymbol{x}_n \in \mathcal{X}_l} \xi_n^\star\right)$
4. **RETURN** $\mathcal{P}_l \in \boldsymbol{\mathcal{P}}$ where $\mathcal{P}_l = \{\boldsymbol{x}_j^{(l)} \in \mathcal{X}_l \,:\, A_j^{(l)} = 1\}$

4.2 Errors & Metrics

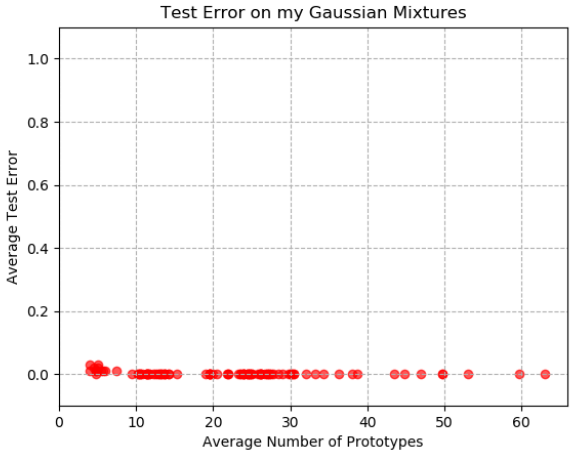
I implemented the two errors that were required: the *test error*, and the *cover error*.

For the **test error**, I simply performed a kNN type of error, where using the newly created prototype set, I found the nearest prototype and assigned the test data point that prototype's label. If that label did not match the true label of the test data point, it is regarded as misclassified.

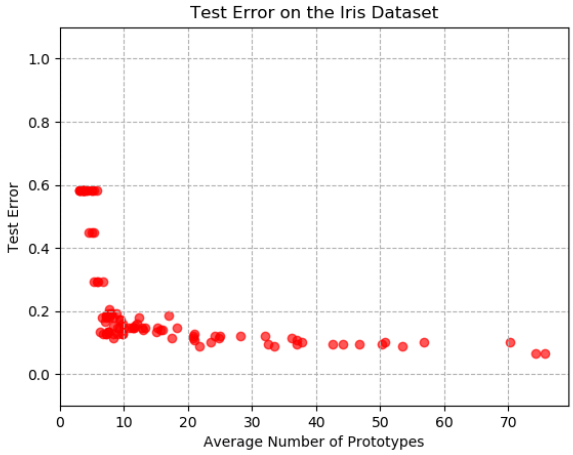
For the **cover error**, I checked what balls covered our test point based off of the prototype set and the *radius* ε of the classifier. If it was only in *one* ball, I gave it that ball's prototype's label (usually, this is also the nearest prototype). If it were in multiple balls, I check first if all the balls covering it belong to the same class. If all the balls covering the test point were of the same class, then the test point is classified as the class of those balls. If the balls that cover the test point are from *multiple* classes, then the test data point is regarded as misclassified. Furthermore, if the test data point is in *no* balls, then it is regarded as misclassified.

5 Results

I first made sure my implementation above works with two "easy" datasets: a Gaussian Mixtures Model that I generated to run basic tests on as I wrote the code, and the Iris Dataset. The results are shown on the next page. Every single run is performed with K-fold cross-validation with K=4.

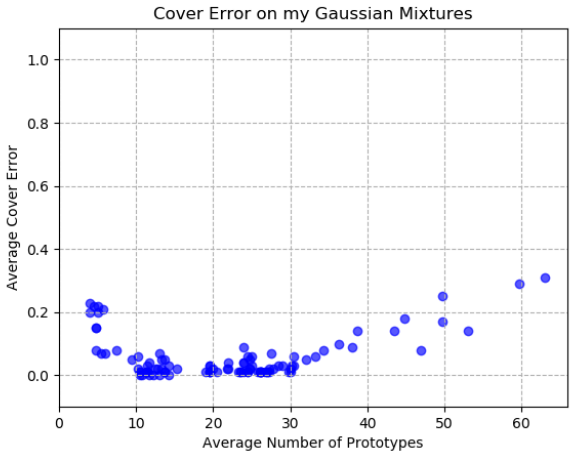


(a) Gaussian Mixtures Test Error

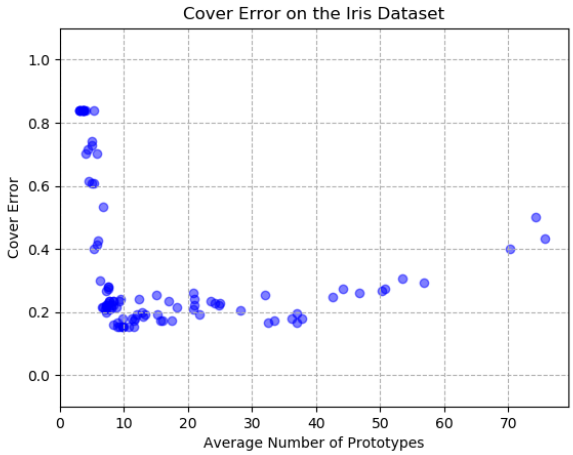


(b) Iris Dataset Test Error

Figure 1: Test Errors on "easy" datasets

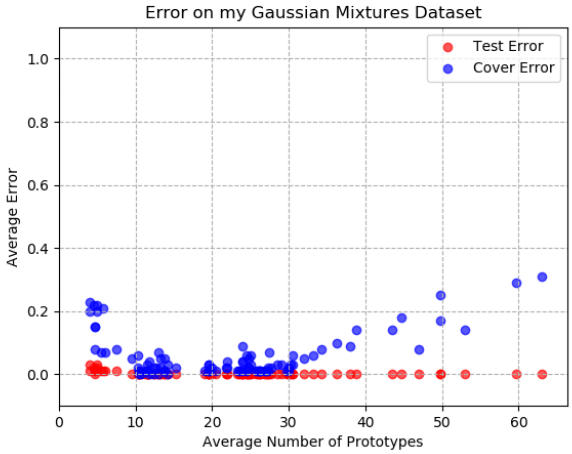


(a) Gaussian Mixtures Test Error

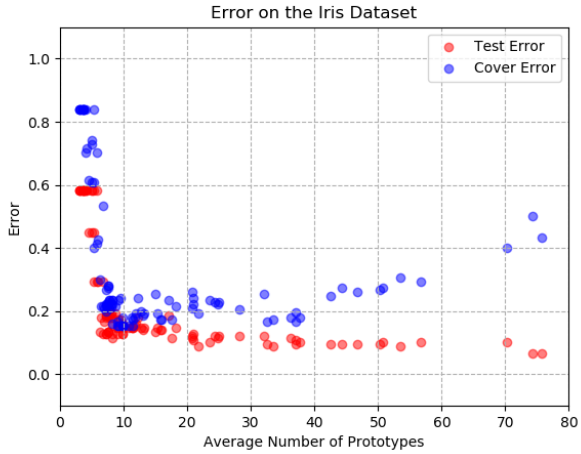


(b) Iris Dataset Cover Error

Figure 2: Cover Errors on "easy" datasets

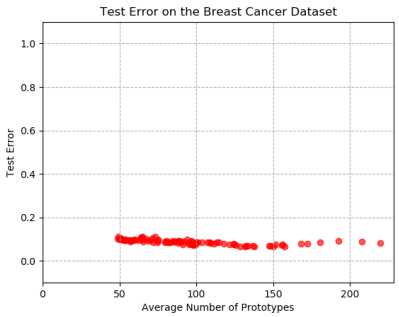


(a) Gaussian Mixtures Error

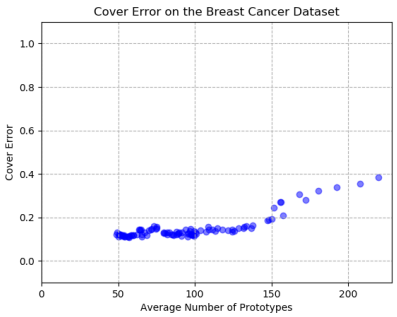


(b) Iris Dataset Error

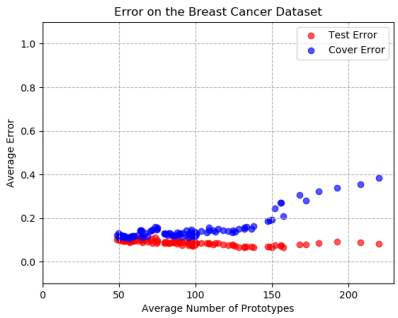
Figure 3: Summary of performance on "easy" datasets



(a) Breast Cancer Dataset Test Error

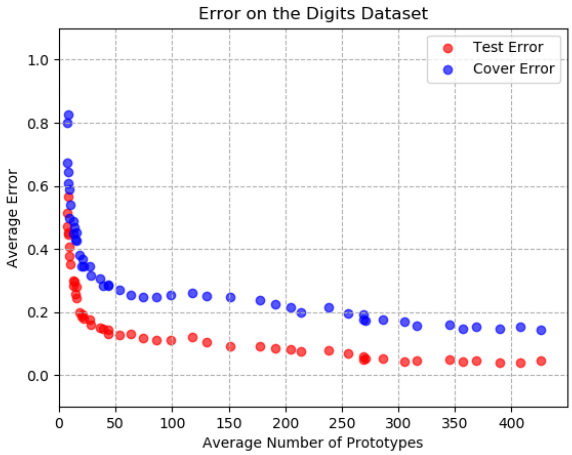


(b) Breast Cancer Dataset Cover Error

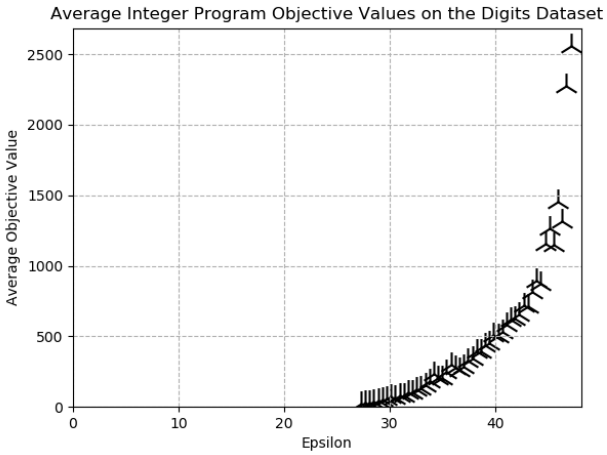


(c) Summary of performance on the Breast Cancer Dataset

Figure 4: Performance on the Breast Cancer Dataset



(a) Summary of performance on the Digits Dataset



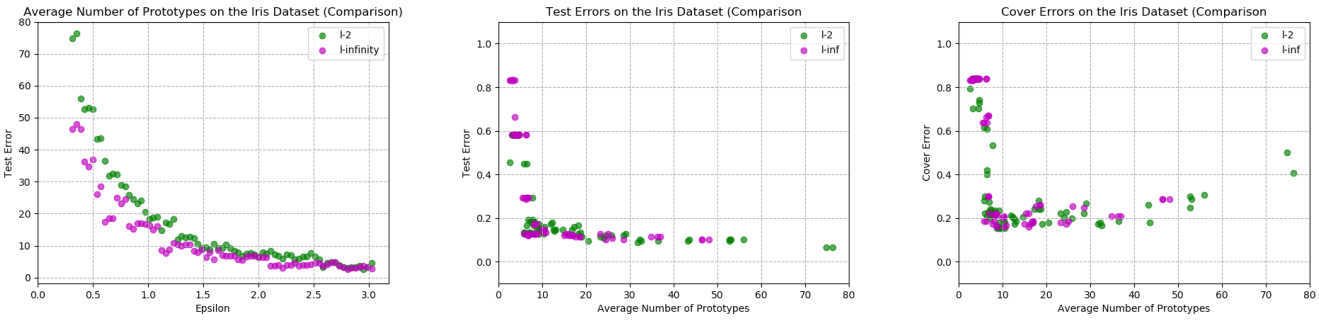
(b) Rounded Integer Program Objective Value for the Digits Dataset

Figure 5: Digits Dataset Analysis

I then analyze the larger Breast Cancer Dataset using the same methods. The results are presented in Figure 4. We see that that it follows the same pattern where increasing model complexity can lead to an increase in error.

We then analyze the optimal value output from the integer program using the Digits Dataset. The results are shown in Figure 5.

Finally, we add a modification to the algorithm, where instead of defining the neighborhood with the $l2$-norm, I define it with the infinity norm. By doing so I hope to capture a larger area with the prototype to define larger neighborhoods with less prototypes. The results are shown in Figure 6.



(a) Iris Dataset Prototype Count (b) Iris Dataset Test Error (c) Iris Dataset Cover Error

Figure 6: Performance on the Iris Dataset

References

[1] Bien, J., Tibshirani, R. (2011) *Prototype Selection for Interpretable Classification*. The Annals of Applied Statistics. **5(4)**, 2403-2424.