OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Heuristic Prototype Selection for Regression

Debraj Basu DBASU@ADOBE.COM
Deepak Pai DPAI@ADOBE.COM
Joshua Sweetkind-Singer SWEETKIN@ADOBE.COM
Adobe Inc., San Jose

Abstract

We propose a prototype method in the regression setting, that leverages proximity measure of samples both in the feature space as well as the label space. A good set of prototypes may substantially reduce the space- and time-complexity costs over using k-nearest-neighbors on the original dataset and can potentially yield both improved test error and improved robustness to outliers. We formulate the problem as a mixed-integer program that captures several properties that are intuitively desirable of a good prototype set. As is typical of such formulations, exact solutions are computationally intractable, and we obtain an approximate solution via an algorithm that successively refines the prototype set by greedily selecting samples from the dataset. Our algorithm easily generalizes to regression problems with vector-valued outputs. It also handles classification problems as a special case, by encoding class labels as one-hot vectors, resulting in prototypes identical to the those yielded by a popular set-cover-based method in the classification setting due to Bien and Tibshirani, 2012. Our experiments demonstrate that the prototypes selected by this method provides better generalization with an average space reduction of 75%, and improved tolerance to outliers.

1. Introduction.

One of the primary motivations behind prototype selection methods is to improve upon the k-nearest-neighbors algorithm. k-NN [9, 12] is a well-known technique for prediction [21], despite its shortcomings: a) high storage and computation costs [10], and b) sensitivity to outliers present in the original training set [21, 29]. To overcome these limitations, prototype-selection methods cull from the original training set a much smaller set of instances, called *prototypes*, and then, typically, run the k-NN algorithm on those. Reducing the dataset affords an additional way to control complexity, by varying the maximum number of prototypes, potentially leading to better generalization and improved tolerance to outliers. All of these advantages have been verified in the classification setting [6–8, 17, 18, 20, 21, 23, 30]. In such settings, prototype methods exploit the separation of samples in the feature space, based on classes they belong to, in order to identify "prototypical" instances of local representative value. Coming up with a method that drives the discovery of such a set of prototypes for regression with vector-valued outputs is the primary focus of this paper.

Our work is most closely related to [8, 24, 28, 31], all of which have employed integer optimization for identifying prototypes in a classification setting. [24, 28] define the *set covering machine* (SCM) for a binary classification problem in which the goal is to identify a minimal disjunction of binary functions of the features. The proposed formulation, resembles that of [8] for a multi-class classification dataset, which is a specialization of the general SCM formulation from [24].

The underlying premise of our method is that a "prototypical" point represents a region in which the training points have similar labels to that of the prototype, while avoiding proximity to any train-

ing point with a highly dissimilar label. We encode these properties into a mixed-integer program and propose greedy heuristics for approximating the solution. Our formulation takes advantage of proximity measures in both the feature and label space to select salient prototypes from a dataset with continuous or discrete valued labels, or, indeed, vectors whose components are a mixture of the two. It also handles classification problems as a special case, by encoding class labels as one-hot vectors, resulting in prototypes identical to those yielded by [8]. (See Section 3 and Appendix E.)

Our experimental findings substantiate the benefits of embedding these properties into our method which results in salient prototypes. We observe lower prediction errors using the k-NN rule for prototypes selected by our method, as compared to several natural baselines, while achieving significant compression in the samples (see Figure 1a and Table 2). Furthermore, as is expected from a good prototype set, our prototypes result in lower test errors even for smaller values of k, unlike some of our baselines (see Figure 5). Beyond these advantages, we see that the prototypes selected by our method are more tolerant to outliers present in the training set (see Table 1).

2. Optimization Problem

Consider a labeled dataset $\mathcal{D} : \{\mathcal{X}, \mathcal{Y}\}$ where $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n \in \mathbb{R}^d$ and $\mathcal{Y} = \{y_i\}_{i=1}^n \in \mathbb{R}$, and n is the total number of points in the dataset. Without loss of generality we assume $y_i \in [0, 1] \, \forall i \in [n]$ where $[n] := \{1, 2, \ldots, n\}$. Let the dissimilarity measure between two points $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ be denoted by $d(\mathbf{x}_i, \mathbf{x}_j)$. For a point $\mathbf{x}_i$ and a given $\epsilon \in \mathbb{R}^+$, let the set $B(\mathbf{x}_i)$ be defined as $\{\mathbf{x} \in \mathbb{R}^d | d(\mathbf{x}, \mathbf{x}_i) \leq \epsilon\}$. Therefore $B(\mathbf{x}_i)$ is the ball of radius ϵ centered at $\mathbf{x}_i$. As indicated in [8], set cover is a useful clustering algorithm that can be used to find the smallest number of ϵ balls, with the corresponding centers as prototypes, that cover $\mathcal{D}$. This is done by solving the following formulation for set-cover optimization, $\min_{\alpha_j} \sum_{j=1}^n \alpha_j \; s.t. \; \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j \geq 1, \alpha_i \in \{0, 1\} \forall i \in [n]$. Here α_i is 1 if $\mathbf{x}_i$ is a prototype, 0 otherwise. $\sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j$ is the number of prototypes whose ϵ balls cover $\mathbf{x}_i$, which must be at least 1. However, when samples belong to discrete domains, as in the classification setting, one could utilize the labels of each point. Similarly, in a regression setting, one could leverage the natural ordering of continuous-valued labels to come up with a more meaningful set of prototypes that covers the dataset.

In a generic sense, we expect to ensure that each point is covered by at least one prototype with a similar label. In addition, we prefer each point to not have any prototype with a widely dissimilar label, covering it. This amounts to prototypes covering points only from the same class as in [8] for classification. By selecting a small set of such prototypes, the dataset is reduced to a few salient points thereby achieving sparsity in the samples. In order to do so, we enlist the following properties, that are desirable of the ϵ-covering balls induced by the selected prototype set: **(a)** Must cover as many points with similar labels, as possible; **(b)** Must not cover points with dissimilar labels; **(c)** Must be sparse (uses a few prototypes). Note that (b) could result in a few points left uncovered.
Formulation: We propose a generic mixed-integer formulation below, in variables $\{\alpha_i\}_{i=1}^n$, $\{\xi_i\}_{i=1}^n$, $\{\eta_i\}_{i=1}^n$ for identifying prototypes from a labeled dataset $\mathcal{D}$, that satisfy the above properties.

$$\min_{\alpha_i, \xi_i, \eta_i} \sum_{i=1}^n \xi_i + \sum_{i=1}^n \eta_i + \lambda \sum_{i=1}^n \alpha_i$$

$$\sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j (1 - |y_i - y_j|) \geq 1 - \xi_i, \quad \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j |y_i - y_j| \leq \eta_i, \; \alpha_i \in \{0, 1\}, \quad \xi_i, \eta_i \geq 0, \forall i \in [n] \tag{1}$$

The binary variables α_i indicate which $\mathbf{x}_i$ are selected as prototypes, while ξ_i and η_i are non-negative slack variables. We use $|y_i - y_j|$ as a dissimilarity measure between the labels of $\mathbf{x}_i$ and $\mathbf{x}_j$. Any other dissimilarity measure could be substituted here.

The first set of constraints in (1) uses $\alpha_j(1-|y_i-y_j|) \in [0,1]$, which is close to 1 if $\mathbf{x}_j$ has been selected as a prototype and has a highly similar label to that of $\mathbf{x}_i$. The sum of these quantities over all prototypes ϵ-close to $\mathbf{x}_i$ must preferably be at least 1, and, when that is not the case, we penalize the prototype set via the slack variable $\xi_i = 1 - \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j(1-|y_i-y_j|)$, which attempts to capture the extent to which $\mathbf{x}_i$ is not completely covered in the label space by all the prototypes that cover it in the feature space. We point out that this formulation considers two prototypes that half-cover $\mathbf{x}_i$ in the label space to be as good as one prototype that fully covers it. This yields decent prototypes, because the overall formulation discourages prototypes from clustering together.

The second set of constraints in (1) uses $\alpha_j|y_i - y_j| \in [0,1]$, which is close to 1 if $\mathbf{x}_j$ has been selected as a prototype and has a highly dissimilar label from that of $\mathbf{x}_i$. The sum of these quantities over all prototypes ϵ-close to $\mathbf{x}_i$ must preferably be 0, and, when that is not the case, we penalize the prototype set via the slack variable $\eta_i = \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j|y_i - y_j|$, which captures the extent to which $\mathbf{x}_i$ is mis-covered in the label space by all the prototypes that cover it in the feature space. The above penalties take into account similarities in both the feature space, by insisting that the prototypes be ϵ-close to the points they represent, and in the label space, by encouraging a point to be represented by prototypes with more similar labels.

Finally we also penalize the total number of selected prototypes, $\sum_i \alpha_i$, in the objective. Note that the feasible set of this formulation comprises all 2^n possible prototype sets. λ is a hyper-parameter that is used to trade off between consistency and sparsity by controlling the number of prototypes selected, see Appendix C for a discussion on the same.

Optimal Solution: Arriving at the optimal solution to our formulation in (1) is provably computationally difficult, as is typical for integer optimization problems such as set cover, which can in fact be reduced to (1), rendering our formulation at least as hard. Typically mixed-integer programs are solved using branch and bound techniques [27], which is infeasible here as the size of the problem is proportional to the number of samples in the dataset. Therefore in Section 2.1 we propose an approximate solution via a greedy algorithm, Algorithm 1.

2.1. Greedy Approach

Our greedy solution can be better understood by an equivalent formulation of (1) in which the objective is given by, $\min_{\alpha_i} \sum_{i=1}^n \max\left\{0, 1-\sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j(1-|y_i-y_j|)\right\} + \sum_{i=1}^n \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j|y_i - y_j| + \lambda \sum_{i=1}^n \alpha_i$ such that $\alpha_i \in \{0,1\} \quad \forall i \in [n]$. Now consider an existing prototype set indexed by $\mathcal{P} \subset [n]$. Our greedy algorithm considers the effect of appending a new point, $\mathbf{x}_k$, to the prototype set, i.e. $\mathcal{P} \leftarrow \mathcal{P} \cup \{k\}$, on the three different terms of the objective,

The last term penalizes the total number of prototypes and increases by λ when a new prototype is selected. The second term can be rewritten as $\sum_{j=1}^n \left[\alpha_j \sum_{i:\mathbf{x}_i \in B(\mathbf{x}_j)} |y_i - y_j|\right]$. Therefore introducing $\mathbf{x}_k$ as a new prototype increases the second term by $\sum_{i:\mathbf{x}_i \in B(\mathbf{x}_k)} |y_i - y_k|$. This can be precomputed in $\mathcal{O}(n^2)$ for all points. Finally, the first term, which is a summation over all data points indexed by $i \in [n]$, is only affected with respect to the data points that lie in the ϵ-ball of $\mathbf{x}_k$. For a point $\mathbf{x}_l$ that does lie in the ball of $\mathbf{x}_k$, the corresponding term in the objective $\max\left\{0, 1 - \sum_{j:\mathbf{x}_l \in B(\mathbf{x}_j)} \alpha_j(1-|y_l-y_j|)\right\}$ remains unaffected if it was previously 0, otherwise it is reduced by $\min\left\{1 - \sum_{j:\mathbf{x}_l \in B(\mathbf{x}_j)} \alpha_j(1-|y_l-y_j|), 1-|y_l-y_k|\right\}$. Combining these we have Algorithm 1, where $\Delta\text{Obj}(\mathbf{x}_k) = \Delta\xi(\mathbf{x}_k) - \Delta\eta(\mathbf{x}_k) - \lambda$ where $\Delta\xi(\mathbf{x}_k) = \sum_{i=1}^n \min\left\{1 - |y_k - y_i|, \max\{0, 1 - \sum_{j:\mathbf{x}_i \in B(\mathbf{x}_j)} \alpha_j(1-|y_i-y_j|)\}\right\}$ and $\Delta\eta(\mathbf{x}_k) = \sum_{j:\mathbf{x}_j \in B(\mathbf{x}_k)} |y_k - y_j|$.

Algorithm 1 Heuristic Prototype Selection

Initialize $\mathcal{P} = \phi$
do
  Find $i' \leftarrow \arg\max_{i \in [n] \setminus \mathcal{P}} \Delta\text{Obj}(\mathbf{x}_i)$
  If $\Delta\text{Obj}(\mathbf{x}_{i'}) > 0$ **then do** $\mathcal{P} \leftarrow \mathcal{P} \cup \{ i' \}$
while $\Delta\text{Obj}(\mathbf{x}_{i'}) > 0$
return $\mathcal{P}$

Our choice of a new prototype guarantees a net reduction in the objective in each iteration. The algorithm terminates when including the next prototype would not decrease the objective. Observe that $\Delta\eta(\mathbf{x}_k)$ can be easily cached for speedup. Let $C(k)$ denote the number of points in the ϵ covering ball of $\mathbf{x}_k$ and p is the cardinality of the prototype set $\mathcal{P}$ in the current iteration, i.e. the number of points which have already been selected as prototypes. The cost of computing $\Delta\text{Obj}(\mathbf{x}_k)$ is $\mathcal{O}(C(k))$. The per iteration time complexity of Algorithm 1 is therefore $\mathcal{O}\left((n-p)\max_k C(k)\right)$ which can be improved through parallelization. For a fixed number of points, $P \leq n$, selected as prototypes, the total cost of computation is $\mathcal{O}\big(nP \max_k C(k)\big)$.

3. Handling Multi-Valued Outputs

Vector-Valued Regression: The formulation in (1) is directly applicable for a single output regression dataset, i.e. $y_i \in \mathbb{R}$ and $\mathbf{x}_i \in \mathbb{R}^d$, for each $i \in [n]$. A natural extension would be in a multi output setting, i.e. $\mathbf{y}_i \in \mathbb{R}^m$ where we can use a dissimilarity measure for multi-dimensional vectors, in the label space. For simplicity we use the Euclidean distance $\|\mathbf{y}_i - \mathbf{y}_j\|_2$ as the dissimilarity measure between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the label space. The generalization of the formulation for vector-valued regression is provided in Appendix D

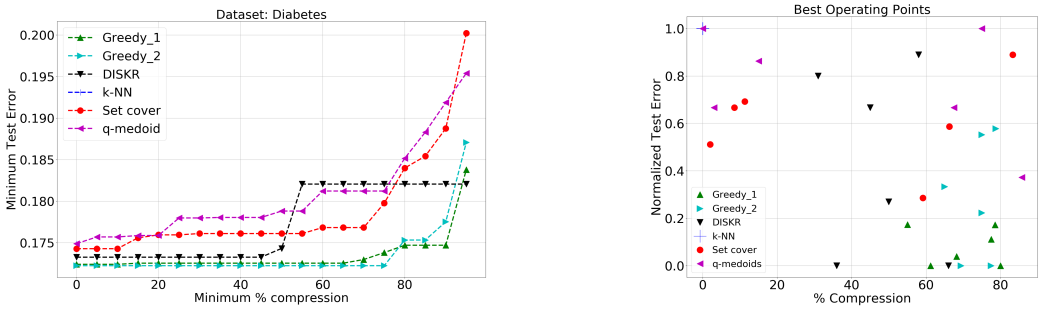
Multi-Class Classification: Our formulation from (1) is directly applicable to a two-class classification problem, where $y_i = 0$ or 1. In fact for two classes, (1) bears similarity to the formulation in [8, Section 2.3], with the difference being that [8] allows a point to be selected as a prototype for a class other than its own, which is unintuitive: we would expect a prototype to be an exemplar of its own class only. In that sense, (1) is a mild improvement over [8], because the set of feasible solutions of (1) does not allow prototypes to represent classes other than their own.

In fact, we can prove that for a multi-class classification problem, Algorithm 1 using either of the greedy criterions from Section 2.1 or Appendix B, is equivalent to the greedy algorithm in [8, Section 3.2], with a modest change mentioned above. The proof is provided in Appendix E. Note that our scheme does not allow multi-membership unlike [8]. This reduces the per iteration complexity of Algorithm 1 by a factor of L, i.e. the total number of classes, as compared to [8].

4. Experiments

We summarize our numerical findings for the regression setting here (the classification setting is already evaluated in [8]) and present them in full with more insights in Appendix F. We conducted experiments on four public datasets and two standard synthetic datasets detailed in Section F.1.1. As such the application of prototype methods for regression remains relatively less explored. Besides k-NN, set cover, and q-medoids which are natural baselines but do not incorporate the label information, we also compare the performance of Algorithm 1 (`Greedy 1` (Section 2.1) and a variant `Greedy 2` (Appendix B)) with a popular prototype method for the regression setting called decremental instance selection for k-NN regression (`DISKR`) [33], which reduces the dataset successively by removing samples based on their marginal contribution to the training error. Figure 1 summa-

rizes our observations on performing extensive cross validation over neighborhood sizes ϵ (resulting in different number of prototypes P), number of nearest neighbors k for each algorithm on every dataset. In our experimental setup, $d(\cdot, \cdot)$ in Section 2 is the Euclidean distance for consistency and simplicity, however this is not necessary and (1) can potentially benefit from using other non-Euclidean dissimilarity metrics [4]. We also set $\lambda = 1/n$, so that the number of prototypes only plays a role when there is a tie between two different solutions.



(*a*) (*b*)

Figure 1 In Figure 1a we compare the best test RMSE given the minimum required % compression of the Diabetes dataset [14]. Algorithm 1 denoted by `Greedy 1` Section 2.1 and `Greedy 2` Appendix B outperforms all other baselines achieving lower RMSE. In Figure 1b we plot the best operating point for each dataset, identified via extensive cross-validation and summarized in Table 2. For each dataset, the best test RMSE for different algorithms is normalized to $[0, 1]$. Algorithm 1 is in the bottom right section as desirable. `DISKR` exhibits low test errors half the time with modest compression. k-NN at the top left corner exhibits poorest performance overall.

We also experimentally evaluate Algorithm 1's tolerance to outliers in Section F.3, by increasingly corrupting the training dataset with Gaussian noise added to the labels, and computing the minimum test error. In Table 1 the robustness of the best two algorithms in each column are in bold. Here the average is taken across different values of k and for percentage of noisy training data between 0% to 50%, for each dataset. Our algorithm appears among the top three for four out of six datasets in Table 1, exhibiting a smaller increase in the minimum RMSE than the other baselines. `DISKR` [33] also provides robust results as it involves a heuristic for the removal of outliers in its first phase.

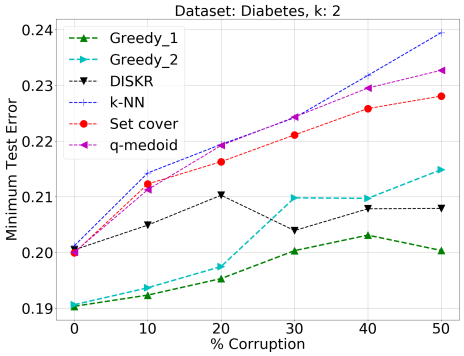


Figure 2 Minimum test RMSE for different % corruption of the Diabetes training dataset for $k = 2$.

	FR1	FR2	DB	IN	WN	WA
`Greedy_1`	4.3	***1.1***	***6.4***	***8.9***	***2.6***	***1.2***
`Greedy_2`	***3.6***	***1.3***	***7.8***	9.0	***2.6***	1.6
`DISKR`	***4.0***	1.5	9.2	***8.4***	4.5	1.5
k-NN	4.1	2.8	13.1	14.9	9.0	1.9
`Set cover`	5.3	1.7	11.6	12.7	4.9	***1.3***
`q-medoids`	4.6	1.8	12.3	9.4	4.9	2.1

Table 1 Mean percentage increase in minimum test RMSE for **(FR1)** Friedman 1, **(FR2)** Friedman 2, **(DB)** Diabetes, **(IN)** Insurance, **(WI)** Wine Quality, **(WA)** Water Quality datasets.

References

[1] Friedman #2 dataset. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_friedman3, . Accessed: 2020-10-01.

[2] Friedman #1 dataset. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_friedman1, . Accessed: 2020-10-01.

[3] Swedish auto insurance dataset. http://college.cengage.com/mathematics/brase/understandable_statistics/7e/students/datasets/slr/frames/slr06.html. Accessed: 2020-02-05.

[4] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*. Springer, 2001.

[5] Álvar Arnaiz-González, José-Francisco Díez-Pastor, Juan José Rodríguez Diez, and César Ignacio García-Osorio. Instance selection for regression: Adapting DROP. *Neurocomputing*, 201:66–81, 2016.

[6] Dimitris Bertsimas and Romy Shioda. Classification and regression via integer optimization. *Operations Research*, 55(2):252–271, 2007.

[7] Binay K. Bhattacharya, Ronald S. Poulsen, and Godfried T. Toussaint. Application of proximity graphs to editing nearest neighbor decision rule. In *International Symposium on Information Theory, 1981.*

[8] Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *Ann. Appl. Stat.*, 5(4):2403–2424, 12 2011. doi: 10.1214/11-AOAS495. URL https://doi.org/10.1214/11-AOAS495.

[9] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[10] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6(2):153–172, Apr 2002.

[11] JA Cornell and RD Berger. Factors that influence the value of the coefficient of determination in simple linear and nonlinear regression models. *Phytopathology*, 77(1), 1987. doi: 10.1094/phyto-77-63.

[12] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, Jan 1967. doi: 10.1109/TIT.1967.1053964.

[13] Y. Dodge. *Statistical data analysis based on the L1-norm and related methods*. Computational statistics & data analysis ; v.6, no.3,4. ISBN 9780444702739.

[14] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[15] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Trans. Systems, Man, and Cybernetics*, 6(4):325–327, 1976.

[16] Sašo Džeroski, Damjan Demšar, and Jasna Grbović. Predicting chemical parameters of river water quality from bioindicator data. *Applied Intelligence*, 13(1):7–17, Jul 2000.

[17] Ehsan Elhamifar, Guillermo Sapiro, and S. Shankar Sastry. Dissimilarity-based sparse subset selection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(11), 2016.

[18] H. A. Fayed and A. F. Atiya. A novel template reduction approach for the k-nearest neighbor method. *IEEE Transactions on Neural Networks*, 20(5):890–896, May 2009. doi: 10.1109/TNN.2009.2018547.

[19] Jerome H. Friedman. Multivariate adaptive regression splines. *Ann. Statist.*, (1):1–67, 03 . doi: 10.1214/aos/1176347963.

[20] Salvador García, José Ramón Cano, and Francisco Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709, 2008.

[21] Salvador García, Joaquín Derrac, José Ramón Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3), 2012.

[22] Mihajlo Grbovic and Slobodan Vucetic. Regression learning vector quantization. In *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, pages 788–793.

[23] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, May 1968. doi: 10.1109/TIT.1968.1054155.

[24] Zakria Hussain, Sandor Szedmak, and John Shawe-Taylor. The linear programming set covering machine. 2004.

[25] Mirosław Kordos and Marcin Blachnik. Instance selection with neural networks for regression problems. In *Artificial Neural Networks and Machine Learning – ICANN 2012*.

[26] B. Li, Y. Wang, H. K. Lee, A. Dempster, and C. Rizos. Method for yielding a database of location fingerprints in wlan. *IEEE Proceedings - Communications*, 152(5), Oct 2005. doi: 10.1049/ip-com:20050078.

[27] Jeff T. Linderoth and Martin W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS J. Comput.*, 11(2):173–187, 1999.

[28] Mario Marchand and John Shawe Taylor. The set covering machine. *J. Mach. Learn. Res.*, 3: 723–746, March 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944952.

[29] Seishi Okamoto and Yugami Nobuhiro. An average-case analysis of the k-nearest neighbor classifier for noisy domains. In *Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'97, San Francisco, CA, USA, 1997.

[30] Elzbieta Pekalska, Robert P. W. Duin, and Pavel Paclík. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189–208, 2006.

[31] Carey E. Priebe, David J. Marchette, Jason G. DeVinney, and Diego A. Socolinsky. Classification using class cover catch digraphs. *Journal of Classification*, 20(1):003–023, May 2003. ISSN 1432-1343.

[32] Diego Furtado Silva and Laboratorio De Intelligencia Computacional. How k-nearest neighbor parameters affect its performance. In *Simposio Argentino de Inteligencia Artificial (ASAI 2009), 95 – 106*, 2009.

[33] Yunsheng Song, Jiye Liang, Jing Lu, and Xingwang Zhao. An efficient instance selection algorithm for k nearest neighbor regression. *Neurocomputing*, 251:26–34, 2017.

Appendix A. Omitted figure from Section 2.1

We numerically demonstrate the efficacy of Algorithm 1 by comparing it with the optimal solution of the LP relaxation of (1) that results from relaxing $\alpha_i \in [0, 1]$ for all $i \in [n]$, see Figure 3.

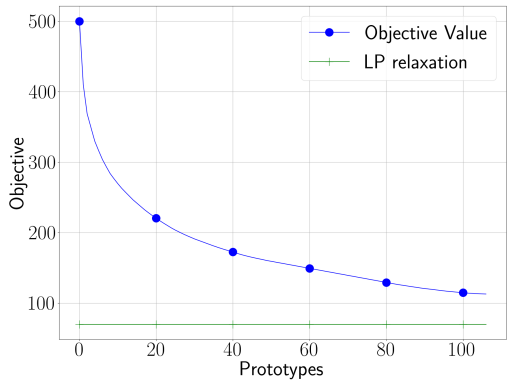


Figure 3 Comparison of the progress of our greedy algorithm for the wine quality dataset [14], with the optimal solution of the LP relaxation. Note that the optimal solution of our formulation is expected to lie in the range between the lowest point of the greedy algorithm and the LP objective value.

Appendix B. Modified Greedy Criterion

We also try another less-than-standard greedy approach for solving (1): in each iteration we select the point $\mathbf{x}_k$ that has the best tradeoff of covering points with similar labels, that remain uncovered, while avoiding covering points with dissimilar labels. For doing so, we define $\text{Set}(\xi_k) := \mathcal{X} \cap (B(\mathbf{x}_k) \setminus \bigcup_{\mathbf{x}_{j'} \in \mathcal{P}} B(\mathbf{x}_{j'}))$ i.e. the set of previously uncovered points that are covered by $\mathbf{x}_k$; $\text{Set}(\eta_k) := \mathcal{X} \cap B(\mathbf{x}_k)$; and $\Delta\text{Obj}(\mathbf{x}_k) = \Delta\xi(\mathbf{x}_k) - \Delta\eta(\mathbf{x}_k) - \lambda$, where $\Delta\xi(\mathbf{x}_k) = \sum_{j:\mathbf{x}_j \in \text{Set}(\xi_k)} \big(1 - |y_j - y_k|\big)$, $\Delta\eta(\mathbf{x}_k) = \sum_{j:\mathbf{x}_j \in \text{Set}(\eta_k)} |y_j - y_k|$. Numerically we observe both the approaches performing similarly, with minor variations in the selected prototypes, and the one in Section 2.1 gaining an upper hand at times, as is expected.

Appendix C. Dataset Condensation

The prototypes chosen by Algorithm 1 adhere to a set of properties, which trade off between consistency (properties (a) and (b) in Section 2), and sparsity (property (c) in Section 2) via the λ parameter. Therefore for smaller values of λ, Algorithm 1 returns a set of prototypes which hold more value in terms of prediction performance, whereas for larger values, it primarily serves as a dataset condensation algorithm.

Well-known condensation schemes such as in [23] aim to preserve accuracy by maintaining decision surfaces of k-NN with a *minimal consistent subset* of the data. This set often comprises of prototypes which lie closer to the decision boundaries, as also in [7, 18]. Our algorithm, on the other hand, selects prototypes greedily based on their ability to represent points in their neighborhood. However, as discussed earlier, for larger values of λ we move our intent of prototype selection from consistency more towards condensation of data.

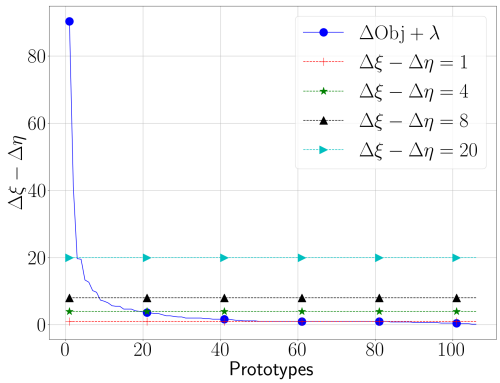


Figure 4 Breakpoints for Algorithm 1 run on wine quality dataset, for a given ϵ in term of different values of λ and the corresponding sparsity induced.

Algorithm 1 iteratively selects a point $\mathbf{x}$ as a prototype based on its $\Delta\text{Obj}(\mathbf{x})$ which is $\Delta\xi(\mathbf{x}) - \Delta\eta(\mathbf{x}) - \lambda$. Here $\Delta\xi(\mathbf{x}) - \Delta\eta(\mathbf{x})$, which can be easily shown to be non-increasing as the algorithm progresses, describes a marginal score of representation while providing a natural ordering of prototypes based on their ability for covering new points in close proximity both in the feature and label space, and not covering points that are further away. λ controls the sparsity without affecting the order of selection of points as prototypes.

In Figure 4, the horizontal lines denote termination points for our algorithm, corresponding to the values that λ takes. 70 prototypes are selected for $\lambda = 1$, whereas for $\lambda = 4, 8$, and 20, only 20, 8 and 2 prototypes are selected respectively. The order in which prototypes are selected remains preserved and the corresponding values on the y-axis demonstrate the strength of representation, for each prototype. However, note that the ordering of prototypes is in fact, an artifact of the greedy algorithm, and the marginal reduction in objective due to each prototype may differ in the optimal solution.

Appendix D. Vector Valued Regression

With $\Delta\mathbf{y}_{\max} := \max_{\mathbf{y}_i,\mathbf{y}_j\in\mathcal{Y}} \|\mathbf{y}_i - \mathbf{y}_j\|_2$, the formulation in (1) can be modified as follows,

$$\min_{\alpha_i,\xi_i,\eta_i} \sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\eta_i + \lambda\sum_{i=1}^{n}\alpha_i$$
$$\sum_{j:\mathbf{x}_i\in B(\mathbf{x}_j)} \alpha_j\left(1 - \frac{\|\mathbf{y}_i-\mathbf{y}_j\|_2}{\Delta\mathbf{y}_{\max}}\right) \geq 1-\xi_i, \quad \sum_{j:\mathbf{x}_i\in B(\mathbf{x}_j)} \alpha_j\frac{\|\mathbf{y}_i-\mathbf{y}_j\|_2}{\Delta\mathbf{y}_{\max}} \leq \eta_i,\ \alpha_i\in\{0,1\}, \quad \xi_i,\eta_i\geq 0,\ \forall i\in[n] \tag{2}$$

The corresponding greedy algorithm from Section 2.1 would remain the same, except that now

$$\Delta\xi(\mathbf{x}_k = \sum_{i=1}^{n}\min\left\{1-\frac{\|\mathbf{y}_k-\mathbf{y}_i\|_2}{\Delta\mathbf{y}_{\max}}, \max\left\{0, 1-\sum_{j:\mathbf{x}_i\in B(\mathbf{x}_j)}\alpha_j\left(1-\frac{\|\mathbf{y}_i-\mathbf{y}_j\|_2}{\Delta\mathbf{y}_{\max}}\right)\right\}\right\}. \tag{3}$$

and $\Delta\eta(\mathbf{x}_k) = \sum_{j:\mathbf{x}_j\in B(\mathbf{x}_k)}\frac{\|\mathbf{y}_k-\mathbf{y}_j\|_2}{\Delta\mathbf{y}_{\max}}$.

Similarly the modified greedy criterion in Appendix B would now be using the following, $\Delta\xi(\mathbf{x}_k) = \sum_{j:\mathbf{x}_j\in\text{Set}(\xi_k)}\left(1-\frac{\|\mathbf{y}_j-\mathbf{y}_k\|_2}{\Delta\mathbf{y}_{\max}}\right)$, $\Delta\eta(\mathbf{x}_k) = \sum_{j:\mathbf{x}_j\in\text{Set}(\eta_k)}\frac{\|\mathbf{y}_j-\mathbf{y}_k\|_2}{\Delta\mathbf{y}_{\max}}$.

Appendix E. Omitted proof from Section 3

We will now show that for a multi-class classification problem, Algorithm 1 is equivalent to the greedy algorithm in [8, Section 3.2], with a modest change mentioned above.

Proof For a general L class classification problem, where $\mathcal{D} := \{\mathbf{x}_i, y_i\}_{i=1}^n$, $\mathbf{x}_i\in\mathbb{R}^d$ and $y_i\in[L]$ for all $i\in[n]$, we can directly apply (2) after one-hot encoding $\mathcal{Y}$. Note that $\Delta\mathbf{y}_{\max} = \sqrt{2}$ here when the dissimilarity measure being used is euclidean distance. We will first consider the greedy criterion proposed in Section 2.1. In [8], $\Delta\xi(\mathbf{x}_k, l)$ denotes the number of points in class l lying in the ϵ radius ball centered at $\mathbf{x}_k$, which remain uncovered by previously chosen prototypes for the same class. In our case, we use

$$\max\left\{0, 1-\sum_{j:\mathbf{x}_i\in B(\mathbf{x}_j)}\alpha_j\left(1-\frac{\|\mathbf{y}_i-\mathbf{y}_j\|_2}{\Delta\mathbf{y}_{\max}}\right)\right\}.$$

which is 0 when $\mathbf{x}_i$ lies in the ϵ ball of at least one prototype belonging to the same class, and 1 otherwise. Therefore the expression for $\Delta\xi(\mathbf{x}_k)$ in (3) is essentially reduced to $\sum_{j:\mathbf{x}_j\in\text{Set}(\xi_k)}\left(1-\frac{\|\mathbf{y}_j-\mathbf{y}_k\|_2}{\Delta\mathbf{y}_{\max}}\right)$ where $\text{Set}(\xi_k)$ is defined in Appendix B which coincidentally also results from our modified greedy criterion. Observe that $1-\frac{\|\mathbf{y}_j-\mathbf{y}_k\|_2}{\Delta\mathbf{y}_{\max}}$ is 1 when $\mathbf{x}_k$ and $\mathbf{x}_j$ are from the same class, and 0 otherwise. This is summed over all points $\mathbf{x}_j$ which are covered by the ϵ ball of $\mathbf{x}_k$ and are uncovered by previously selected prototypes. Therefore we have shown that $\Delta\xi(\mathbf{x}_k)$ used in Algorithm 1 with either of our greedy criterions is equivalent to $\Delta\xi(\mathbf{x}_k, l)$ in [8] for $l = \mathbf{y}_k$.

Similarly, $\Delta\eta(\mathbf{x}_k, l)$ in [8] denotes the number of data points which belong to a class that is different from l, and are being covered by the ϵ ball of $\mathbf{x}_k$. Analogously, $\Delta\eta(\mathbf{x}_k)$ in our algorithm uses $\frac{\|\mathbf{y}_j-\mathbf{y}_k\|_2}{\Delta\mathbf{y}_{\max}}$ which is 0 when $\mathbf{x}_k$ and $\mathbf{x}_j$ are from the same class, and 1 otherwise. Since this is summed over all points $\mathbf{x}_j$ which lie in the ϵ ball of $\mathbf{x}_k$, it is equivalent to $\Delta\eta(\mathbf{x}_k, l)$ from [8] for $l = \mathbf{y}_k$. Therefore we observe that our scheme does not allow multi-membership. ■

Appendix F. Omitted Experiments from Section 4

We provide extensive experiment results in a regression setting, demonstrating the efficacy of the prototypes selected by our method. As such, Algorithm 1 handles classification as a special case resulting in prototypes identical to [8] as discussed in Section 3 under Multi-Class Classification, which is why we restrict our experiments to regression. Along the lines of [5, 8, 21, 22, 25, 33], we evaluate our method by comparing its prediction performance, tolerance to outliers and goodness of fit with other natural baselines.

F.1. Experiment Setup

F.1.1. DATASETS

We performed experiments on both synthetic and several public datasets, all of which have continuous valued outputs. The two synthetic datasets with multi-dimensional inputs are as mentioned in [19, Section 4.3, 4.4], also available at [1, 2], comprising 100 samples each labeled using non linear functions of the inputs, with additive gaussian noise drawn from $\mathcal{N}(0, 0.25)$, and for which a good prototype set would possibly be more informative than a linear model.

We also performed experiments on the following datasets, **(a)** Diabetes dataset [14]; **(b)** Wine quality dataset [14]; **(c)** Swedish auto insurance dataset [3]; **(d)** Water quality dataset [16]. Datasets **(a-c)** have single output labels, whereas **(d)** consists of multi-output labels comprising 14 different targets. Our datasets contain both categoric and real-valued features; however, the labels are either all continuous-valued scalars, such as in Datasets **(a-c)**, or vectors, as in **(d)**.

F.1.2. METHODOLOGY AND BASELINES

In our experiments, $d(\cdot, \cdot)$ is Euclidean distance for simplicity, λ is set to $1/n$, and we evaluate Algorithm 1 based on the root mean squared errors (RMSE) obtained when applying it to training and test sets. The latter is handled via five-fold cross-validation for 100 uniformly-spaced values of ϵ between the minimum and maximum inter-point distance.

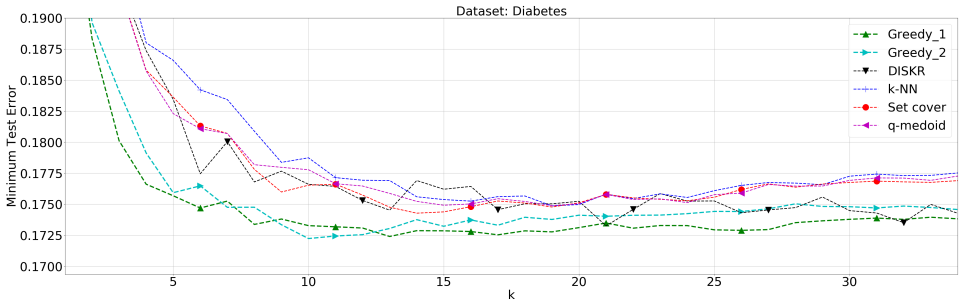
We compare the performance with four natural baselines, viz: for each epsilon we select prototypes using Algorithm 1 with the corresponding greedy criterions, `Greedy_1` from Section 2.1 and `Greedy_2` from Appendix B. Each of these is compared with **(a)** prototypes selected using DISKR algorithm [33], which greedily eliminates samples from the dataset based on whether the presence or absence of the sample enhances the prediction performance of k-NN regression; **(b)** prototypes selected using a greedy algorithm for a simple `set cover` formulation, i.e. running Algorithm 1 with all labels being set as identical, which is equivalent to unsupervised clustering; **(c)** prototypes selected using the q-`medoids` algorithm which selects prototypes that belong to the training data, [1]; **(d)** k-NN algorithm, in which the entire training data is retained as the model for prediction using k-NN.

F.1.3. IMPLEMENTATION

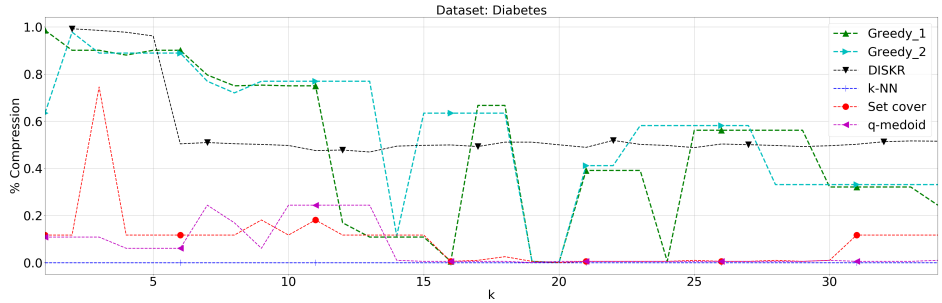
Note that in our implementation of the prediction routine for these schemes, where the k nearest neighbors from the training data, or the k nearest prototypes selected are used, we perform a

1. We refrain from using k here in order to avoid confusion with k-NN. Here q is being used for the number of predefined cluster centers of k-medoids [13] algorithm.

weighted average of the labels, the weights being the inverse of the distance between the test point and the corresponding neighbor as prescribed in [15, 26, 32]. Furthermore, as is standard for nearest-neighbors-based algorithms, we standardize each feature to zero mean and unit variance. Similarly for experiments pertaining to vector-valued outputs, we standardize the regression outputs along each dimension.



(*a*) Minimum test RMSE plotted against k, where k corresponds to the number of nearest prototypes being used for making predictions. For each k, we can generate a plot such as Figure 6 and find the best operating point i.e. point with the lowest test error.



(*b*) The amount of compression achieved by each algorithm for each value of k in Figure 5a in order to achieve the corresponding test RMSE.

Figure 5 Prediction performance on diabetes dataset. Figure 5a and Figure 5b must be studied simultaneously. We see that for lower values of k i.e. for example $k \leq 10$, Algorithm 1 (`Greedy_1` and `Greedy_2`) demonstrates significantly lower test errors as compared to other baselines, while maintaining consistently high dataset compression. This is indicative of a set of good prototypes, for which one would expect to have to use lesser nearest neighbors for achieving good generalization. Our algorithm takes advantage of the label information for choosing smaller set of salient prototypes which is sufficient for regression. We also note that `DISKR` achieves it's lowest test error at $k = 21$, often outperforming other baselines which do not use the label information, for $k \geq 21$.. `Set cover` and `q-medoid` exhibit low test errors for k in the range of 14 to 23, but with consistently low compression.

F.2. Prediction Performance

Figure 6 is a sample cross-validation plot for the diabetes dataset in which the selected prototypes are used for generating predictions based on the 2-nearest neighbor rule. For each algorithm in this figure, we can find the sweet spot in the number of prototypes, that exhibits the lowest test error. Clearly, both `Greedy_1` and `Greedy_2` achieve lower test errors than all of the other baselines

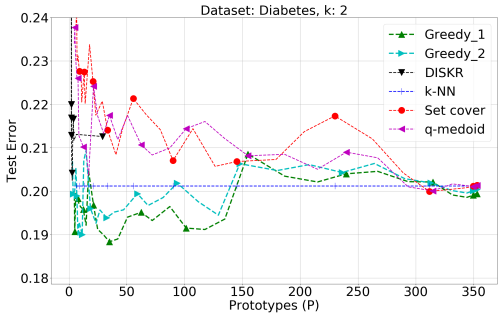


Figure 6 Sample cross-validation plot of test error vs the number of selected prototypes (P) by different algorithms from the Diabetes dataset. The prototypes selected by each algorithm are used for making predictions using the k-nearest neighbors rule. Clearly Algorithm 1 (`Greedy_1` and `Greedy_2`) stands out as the superior one for most P. `DISKR` is a close second at $P = 4$, however its performance is worse than k-NN much alike the other baselines for other values of P. These numerics are generated for all datasets and for a wide range of k-values.

with a highly compressed dataset. Furthermore, as expected, we observe `DISKR` exhibiting competitive test error as compared to k-NN for very small number of prototypes. This is because both, our algorithm, and `DISKR` take advantage of the label information in the process of selecting prototypes (or reducing the dataset). Note that all algorithms except `DISKR` have a setting in which the prototype set equals the entire dataset i.e. no compression and therefore in the worst case the minimum test error for each of them will be at most as much as that of k-NN. As for `DISKR`, it naturally almost always ends up selecting only a fraction of the complete dataset as prototypes and might therefore exhibit worse test errors than k-NN at times.

We repeat this process of finding the best operating point for a wide range of values of k for each dataset, which results in Figure 5 for the diabetes dataset. We observe that our algorithm combines the benefits of utilizing proximity measures in the label space, together with requiring selected prototypes to conform to a set of desirable properties, clearly outperforming cases where the label information is not used. `DISKR` on the other hand exhibits competitive performance for higher values of k, with consistently modest compression.

Our findings about the prediction performance for each dataset, are summarized in Table 2, where columns correspond to datasets, and the rows correspond to different prototype selection schemes. Each cell block, corresponds to the best operating point by test RMSE, for each pair of dataset and algorithm, which is obtained from plots analogous to Figure 5. The two best algorithms by test RMSE are highlighted in each column, in which our algorithm appears with majority, while exhibiting consistently high compression. `DISKR` leverages the label information to appear among the top 3 algorithms, for 4 out of the 6 datasets. Figure 8 provides a concise visualization of the same, where we see that our algorithm resides in the bottom right of the figure exhibiting lower test errors and higher compression across different datasets. On average we incur a factor of 18% reduction in the test RMSE across all datasets by using Algorithm 1 instead of k-NN.

Finally in Figure 7, we observe the superiority of our algorithm in terms of the minimum achievable test error over different values of k, for a minimum required percentage compression of the training data. `DISKR` exhibits competitive performance for compression less than 50%. k-NN exists as a single point with no compression. We also observe these trends for other datasets.

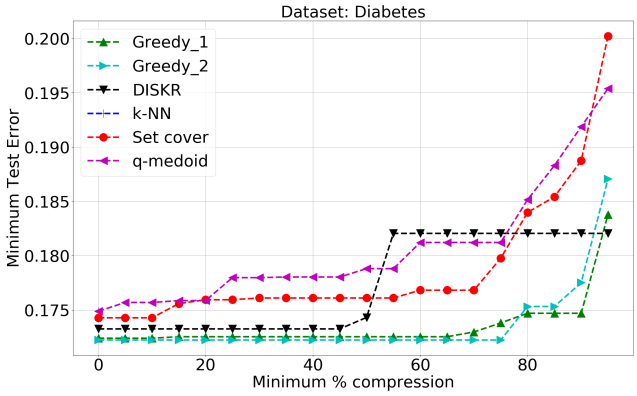


Figure 7 The minimum test RMSE is plotted against the minimum required compression and therefore every plot is non-decreasing. Clearly Algorithm 1 (`Greedy_1` and `Greedy_2`) outperforms all other baselines achieving lower test errors for both low as well as high compression requirements. k-NN is a single point at $(0, 0.175)$.

		FR1	FR2	DB	IN	WN	WA
`Greedy_1`	(1)	**0.150**	**0.187**	**0.1725**	**0.113**	**0.1340**	**0.0901**
	(2)	**55**	**61.25**	**68.25**	**80**	**77.5**	**78.57**
`Greedy_2`	(1)	0.161	**0.188**	**0.1724**	**0.115**	0.1361	**0.0895**
	(2)	75	**65**	**77.46**	**75**	78.75	**69.39**
`DISKR`	(1)	**0.145**	0.189	0.1731	0.121	**0.1335**	0.0923
	(2)	**36**	45	45	58	**66**	31
k-NN	(1)	0.174	0.190	0.1750	0.122	0.1380	0.0930
	(2)	0	0	0	0	0	0
`Set cover`	(1)	0.162	0.189	0.1742	0.121	0.1358	0.0905
	(2)	66.25	8.5	11.26	83.33	2	59.18
`q-medoids`	(1)	0.170	0.189	0.1750	0.122	0.1365	0.0908
	(2)	15	3	0	75	67.5	85.71

Table 2 Summary of Prediction Performance Results (test RMSE) where each column corresponds to a dataset, **(FR1)** Friedman 1, **(FR2)** Friedman 2, **(DB)** Diabetes, **(IN)** Insurance, **(WI)** Wine Quality, **(WA)** Water Quality, and **(1)** Best test error, **(2)** % Compression. For each dataset, we report the best operating point chosen by cross-validating over a range of values for k and number of prototypes (P) selected by each algorithm. Here, the two best algorithms for each column, by test RMSE, are in bold. Observe that both `Greedy_1` and `Greedy_2` exhibit low test errors at consistently high data compression as is expected of a set of good prototypes. This can also be visualized via Figure 8.

F.3. Robustness

As discussed in [21, 29], k-NN methods can suffer from lack of tolerance to outliers because they consider the entire dataset relevant, even when it contains spurious data points. Prototype selection is one way to possibly ameliorate the problem, and in this section we experimentally evaluate Algorithm 1's tolerance to noise vis a vis the other baselines in the following way. We corrupt a dataset by adding gaussian noise to the labels of q% of the training points. We do this for several different values of q between 0% and 50%. The noise is drawn $\mathcal{N}(0, \sigma^2 \mathbb{I}_m)$, where m is the number of target variables and σ^2 is the variance in the labels of the entire dataset, with $\sigma = 1$ for $m > 1$ due to standardization of the labels.

Cross-validation is performed with noisy training data and pure test data as follows. For a given (4/5, 1/5) split of the data into training set and test set respectively, we introduce gaussian noise

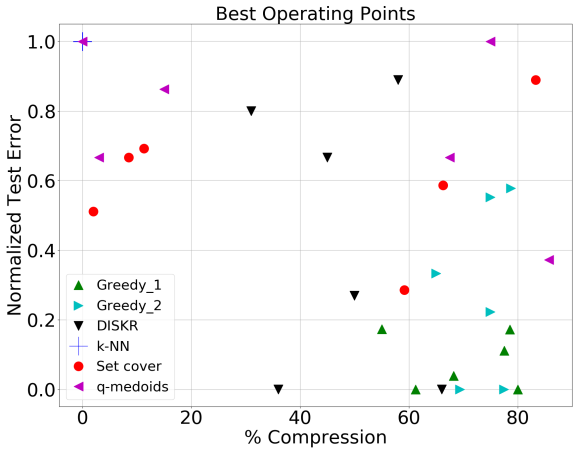


Figure 8 This plot is a visualization for Table 2, where each point corresponds to a cell of Table 2, i.e. a unique pair of dataset and prototype selection algorithm. The test errors in each column of Table 2 are normalized between 0 and 1 and plotted against the percentage compression. Being in the bottom right is most desirable with lower test errors and higher compression. We see that Algorithm 1 (`Greedy_1` and `Greedy_2`) is consistently in the bottom right. The next best algorithm here is `DISKR` which exhibits low test errors half the time, with modest to high compression. k-NN which always resides in the top left has the worst operating points for all the datasets.

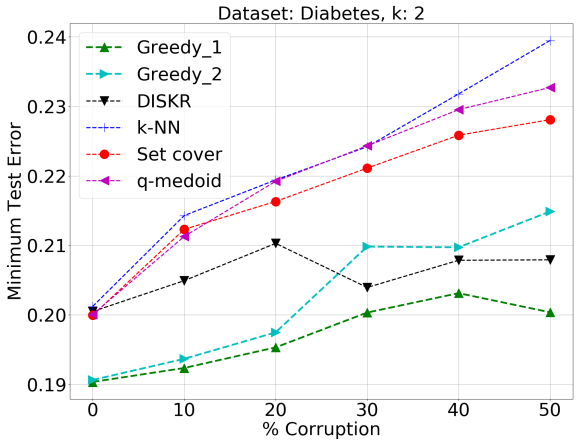


Figure 9 In this plot we demonstrate the effect of corrupting 0 to 50% of the Diabetes dataset, on the performance of our Algorithm and other baselines (in terms of the minimum test RMSE), for a fixed value of $k = 2$. From this plot we observe that the effect of data corruption is more pronounced on our baselines which disregard label information, as compared to on our Algorithm. This experiment is repeated for a wide range of values of k and for every dataset, the result of which are aggregated and summarized in Table 3.

to labels of the training set, but not the test set, and then compute the test error on the test set using the prototypes that were selected from the noisy training set.

In Figure 9, we observe an increasing trend in the minimum test error on increasing the amount of corrupted data for the diabetes dataset, where the predictions for every scheme are generated using 2-nearest prototypes. The increase in test error is most pronounced for k-NN which utilizes the entire dataset. Our algorithm (`Greedy_1` and `Greedy_2`) exhibits a smaller increase in the test error as compared to both `q-medoid` and `Set cover`, both of which do not take advantage of the label information and are therefore affected by data corruption. By the same reasoning, the

	FR1	FR2	DB	IN	WN	WA
`Greedy_1`	4.3	***1.1***	***6.4***	***8.9***	***2.6***	***1.2***
`Greedy_2`	***3.6***	***1.3***	***7.8***	9.0	***2.6***	1.6
`DISKR`	***4.0***	1.5	9.2	***8.4***	4.5	1.5
k-NN	4.1	2.8	13.1	14.9	9.0	1.9
`Set cover`	5.3	1.7	11.6	12.7	4.9	***1.3***
`q-medoids`	4.6	1.8	12.3	9.4	4.9	2.1

Table 3 Mean percentage increase in minimum test RMSE (%): Summary of Results depicting robustness where **(FR1)** Friedman 1, **(FR2)** Friedman 2, **(DB)** Diabetes, **(IN)** Insurance, **(WI)** Wine Quality, **(WA)** Water Quality. The percentage robustness of the best two algorithms in each column are in bold. Here the average is taken across different values of k and for percentage of noisy training data between 0% to 50%, for each dataset. Our algorithm, Algorithm 1, is among the top three for 4 out of 6 datasets.

test error with `DISKR` is the least affected as it not only utilizes the label information, but also removes outliers in the its first phase.

Table 3 reports the average percentage increase in the minimum test error due to data corruption, for different datasets. Observe that *k*-NN is the least robust algorithm for 4 out of the 6 datasets, which is attributed to its lack of tolerance to outliers. On the other hand, both `Greedy_1` and `Greedy_2`, which are variations of Algorithm 1, consistently appear among the top two robust algorithms for 5 and 4 datasets respectively.

F.4. Supervised Clustering

The coefficient of determination (R^2 value as defined in [11]), which represents the percentage of variance in the dependent variables that is explained by the independent variables, is used to evaluate Algorithm 1 as a supervised clustering algorithm in Figure 10. Given a prototype set, each prototype serves as a cluster representative and all samples in the dataset are associated with the cluster corresponding to the nearest prototype. This clustering is evaluated by computing the R^2 value using the label of each prototype as a prediction for each sample in its cluster, which is then compared with the actual label of each sample.

Figure 10 demonstrates the variation of R^2 with the number of clusters for three different datasets. We clearly observe the superiority of our algorithm (`Greedy_1` and `Greedy_2`) for all number of clusters. This is because, the predictions made by looking at the nearest prototype are expected to be more accurate with a good prototype set. For those algorithms which emit an inferior prototype set, extensive cross validation is critical for selecting an appropriate number of nearest neighbors, and such schemes cannot be used for supervised clustering.

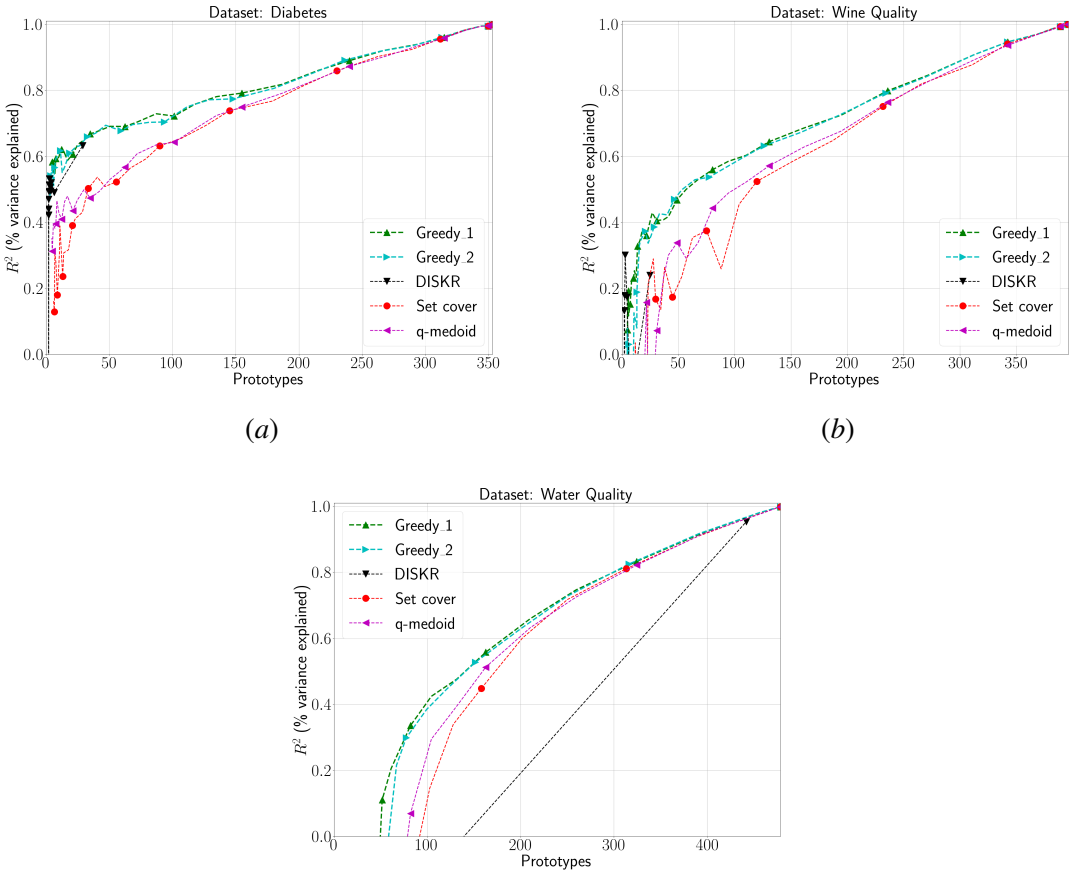


Figure 10 Quantitative measure of supervised clustering performance. Each prototype is the representative of a cluster or group and each group predicts the label of the representative point. If every point is selected as a prototype, then the R^2 value will be 1. As the number of prototypes or groups increases, the fraction of output variance explained also increases. Both `Greedy_1` and `Greedy_2` result in groups which explain a higher fraction of the output variance. The superiority in R^2 values of our Algorithm over our baselines is more pronounced for smaller number of prototypes. `DISKR` naturally selects only a fraction of the total number of samples at most, which is why it does not achieve $R^2 = 1$.