Take home final

SDS385

Fall 2019

This exam has two theoretical and three coding questions. You need to answer all questions.
Read each question carefully, **show your work** and clearly present your answers. Note, the exam is printed two-sided - please don't forget the problems on the even pages!
Please make sure these are your own answers. You can talk to the instructor or the TA if you have any questions.
Turn in your solution via canvas by Dec 16th 10am.

Good Luck!

Name: **Anurag Patil**

UTeid: **arp3969**

1 Experimental questions (30 points)

1. (10 pts) **LSH for Project Gutenberg Data**: Download **Gutenberg.zip** which contains 3036 e-books from project Gutenberg (a small subset of their full corpus) in separate **.txt** files. Convert each book into its set of 2-shingles using scikit-learn's **HashVectorizer** with 2^{18} features. The output will be a **scipy.sparse** matrix X. For Jaccard similarity, use a binary (present/absent) representation instead of shingle counts or frequencies. For each document, generate the MinHash signatures for $n = 90$ hash functions of the same form as in your homework, using a prime p larger than 2^{18}. Use the LSH procedure with $n = 90$ signatures and $b = 30$ bands. Rather than matching signatures explicitly, hash the band of signatures with a separate hash table for each band (see "Mining of Massive Datasets" section 3.4.1). Let any two documents with the same signature key for at least one band be called *neighbors*.

   (a) Approximate the pairwise Jaccard similarity as the fraction of total matching MinHash signatures between all *neighboring* documents (notice how many comparisons this saves vs. $O(3036^2)$). Then find the k most similar document pairs according to this approximation. For $k = 10, 20, 30$, show your results in a table. Specifically, show the document pairs, their true Jaccard score. Hint: You may find it helpful while developing your code to serialize your partial computations (the shingle matrix, signature matrix, etc.) using **cPickle** or **numpy.save**.

   Ans:

   For k = 30 the results are as follows:

Document Pairs	Approximate Jaccard	True Jaccard
2540, 2541	1.0	1.0
410,2761	1.0	1.0
214, 264	1.0	1.0
2012,2700	0.97	0.978
816,1580	0.97	0.978
1469, 2883	0.97	0.978
1019, 1692	0.96	0.95
15282, 2610	0.96	0.93
214, 2970	0.95	0.93
874, 1377	0.95	0.91
1527, 2488	0.95	0.91
264, 2970	0.94	0.91
553, 60	0.93	0.91
392, 2590	0.93	0.894
1575, 2542	0.93	0.8556
313, 867	0.93	0.8556
2178, 2323	0.92	0.85567
313, 1107	0.92	0.85
825, 1027	0.92	0.85
867, 851	0.9	0.83
751, 2271	0.88	0.83
1285, 2228	0.88	0.81
1624, 2037	0.87	0.81
867, 736	0.87	0.81
313, 736	0.86	0.81
465, 2690	0.86	0.8
1107, 2037	0.85	0.8
736, 2716	0.85	0.8
755, 2646	0.84	0.8
313, 2037	0.83	0.8

Thus, it can be seen that for getting the as the true Jaccard similarity decrease the approximation does not work well.

(b) How long will exact search take for $k = 10, 20$ and 30? Compare the search results from LSH and exact search. Also compare the time taken by both for $k = 10, 20$ and 30.

Ans: To get the k=30 similar documents i.e to compute their true Jaccard and Approximate Jaccard it took about 2 mins. To compute the neighbours

it took about 12 mins.

For the exact serach, if we start from the shingles instead of the minhash signatures then it takes about 25 hrs to make all the $O(book^2)$ comparisons. If we start with the minhash signatures for the exact search then it takes about 3 mins to get their True and approximate Jaccard values. The exact search also return the same set of books. The results can be found in the attached python notebook.

2. **Stochastic Variance Reduced Gradient Descent (SVRG).** (10 pts) We are going to implement SVRG which we saw in one of the class presentations. Consider minimizing a function of the following form:

$$F(\beta) = \frac{1}{n}\sum_{i=1}^{n} f(x_i, \beta).$$

A lot of objective functions defined over i.i.d data points, like least squares or maximum likelihood has this form. f is typically convex, and may or may not be differentiable.

There are several possible choices of a stochastic subgradient. Here are two different ones, along with the names they are most known by:

- SGD $\quad \tilde{g}(\beta) = \nabla f_i(\beta)$, $i \in [n]$ random
- SVRG $\quad \tilde{g}(\beta) = \nabla f_i(\beta) + \nabla f_i(\beta^{\text{old}}) - \nabla F(\beta^{\text{old}})$, $i \in [n]$ random

You have already seen "SGD" (aka stochastic gradient descent). We have read about "SVRG" (stochastic variance reduced gradient descent). β^{old} is some old point which we now define. The SVRG algorithm is as follows:

(a) Initialize at some $\boldsymbol{y_1}$, and set $\boldsymbol{x}_1 = \boldsymbol{y}_1$.

(b) Outer loop: For $s = 1, 2, ...$, compute $\mu_s = \nabla F(\boldsymbol{y_s})$, and set $\boldsymbol{x_0} = \boldsymbol{y_s}$.

(c) Inner loop: For $k = 1, ..., m$, run stochastic gradient descent with a re-centered stochastic gradient:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta(\nabla f_I(\boldsymbol{x}_k) - \nabla f_I(\boldsymbol{y}_s) + \mu_s).$$

(d) Option I: set

$$\boldsymbol{y}_{s+1} = \boldsymbol{x}_{m+1}.$$

(e) Option II: set

$$\boldsymbol{y}_{s+1} = \frac{1}{m}\sum_{k=1}^{m} \boldsymbol{x}_{k+1}.$$

(f) Option III: Choose a random index $J \in [m]$, and set

$$\boldsymbol{y}_{s+1} = \boldsymbol{x}_{J+1}.$$

In practice, Option I seems the most natural, though some of the standard proofs rely on other versions. This algorithm explains what β^{old} means. The rationale is that, the re-centered stochastic gradient has a significantly reduced variance.

Recall the multiclass logistic regression problem from HW1. You will again work on the digits data and the news data. For each of these, compare (plot) the performance of GD, SGD, and SVRG for logistic regression **with ℓ^2 regularization**. Plot your results against the number of gradient evaluations for a single training example (GD performs n such evaluations per iteration and SGD performs 1).

Set the regularizer to $1/n$, where n is the number of data points. How does the choice of m affect the performance of SVRG? What seems to be a good choice of m? Now choose a bigger regularizer. What do you observe?

Do option I, II and III perform differently? Show plots to compare their performance on the two datasets.

Ans: **Performance:** As can be seen from the figure, SVRG performs the best for the news and the digits dataset for an alpha of 0.002 for both the datasets. For the digits dataset the variance of SGD is clearly reduced. This is because of considering more amount of history as compared to SGD. Gradient descent uses the enitre history to make one update. The calculations with the old β on the batch and with random sampling adds in great amount of previous information which results in lesser overall variance.

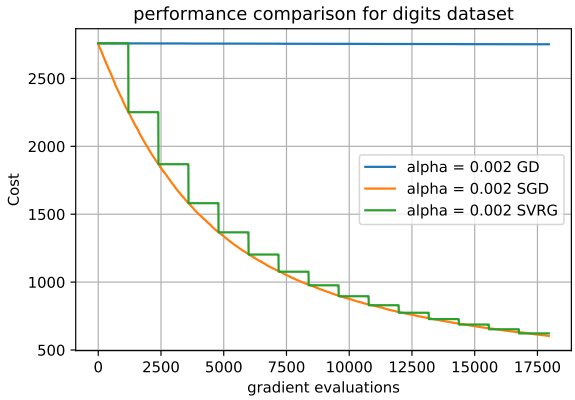

Figure 3: Performance with gradient evaluations for the digits dataset

Gradient Evaluations: As can be seen from the figure, the amount of gradient calculations for one update in direction is very high for gradient descent and the

least for SGD. For SVRG, the number of gradient evaluations are high as compared to SGD, but the amount by which the cost decreases as compared to GD is very close to that of SGD. Hence, the greater number of gradient evalutions are adding value here as compared to gradient descent which also uses histroy to make an update, but suffers from redundant calculations. The direction is retained much better because of that. For the digits dataset the same observation was made.

Different options of SVRG As can be seen from the figure, the option 2 seems

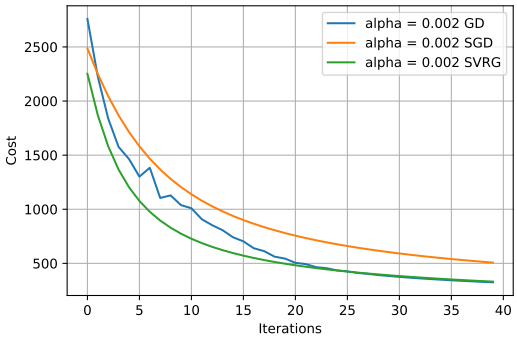

Figure 1: Performance for different method for the Digits dataset

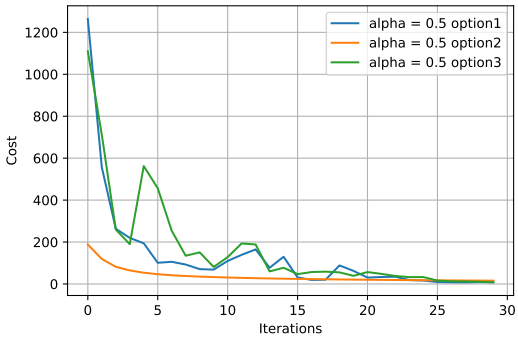

Figure 2: Performance for different options of SVRG for the Digits dataset

to be performing the best. The option 1 uses just the last value to make the update while the option 3 does this randomly. Hence both of them are expected to have more variance in them for decrease in cost. This analysis is shown for the Digits dataset and the news dataset.

The choice of batchsize and regularizer: As can be seen from the figure, With a smaller batchsize the performance of SVRG decreases. But even with half the batchsize the performance drop is very small. This implies that SVRG scales well for smaller batches uptill a certain point. This ofcourse will depend upon the nature of dataset as well. Here, half the batchsize seems to be a good choice. For the regularizer, as it can be seen the performance doesnt change much for about 100 times the value. Thereafter the performance start to decrease. A similar observation was made for the news dataset.

3. **Spectral Clustering on large sparse graphs.** In class we have learned about Spectral Clustering for datasets. In this question you will do Spectral Clustering for sparse networks. In network.txt find a sparse representation of a network in (src, dest, weight) format. Note that the weight is always 1. Build an undirected and unweighted adjacency matrix from this. An adjacency matrix of a graph A is

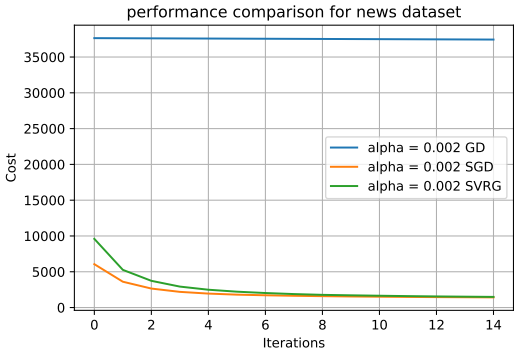

Figure 4: Performance for different method for the News dataset

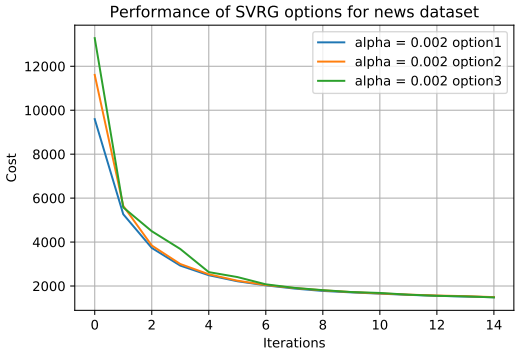

Figure 5: Performance for different options of SVRG for the News dataset

an n by n matrix (n is the number of nodes). $A_{ij} = 1$ if there is an edge between nodes i and j. The true labels of the nodes can be found in comm.txt.

(a) Do Spectral Clustering on this adjacency matrix using your own eigenvector routine. Report the accuracy.

Ans: The accuracy observed is 51 percent. Please refer to the attached spectral clustering notebook for the code.

(b) Open the paper on network clustering by Amini et al from **here**. Read section 2.3.2 to better understand why Spectral Clustering sometimes does not perform well on very sparse networks. Now implement the regularized spectral clustering algorithm proposed in this paper using your own eigenvector routine. You should code up your eigenvector computation such that you do not need to create a $n \times n$ dense matrix. Report the time and accuracy.

Ans: With the regularized implementation the accuracy goes up to 57 percent when only the top two eigen vectors are considered. As more number of eigen vectors are considered the accuracy drastically increase. For the top 5 eigenvectors with 10000 iterations of power method for every eigen vector the accuracy observed was 81 percent. please see the notebook for implementation.

For implementation the idea was to break down evrey calculation such that only vectorxvector or vectorx matrix are done. Also the fact that the perturbation matrix will be filled with only the constants.

2 Theory questions (30 points)

1. Consider the sub-gradient descent algorithm you learned in class (see lecture4). In this question we will work on obtaining rates of convergence for this algorithm. Assume that you are minimizing a convex function f with minimizer x^*. Also assume that f is Lipschitz continuous, i.e. all sub-gradientsare bounded above:

$$\|g\|_2 \leq G.$$

Also assume that the distance of the initial point from the truth x^* is bounded as below:

$$\|x^{(1)} - x^*\|_2 \leq R.$$

Define $d_k := \|x^{(k)} - x^*\|_2$.

(a) Prove that

$$d_{k+1}^2 \leq d_k^2 - 2\alpha_k(f(x^{(k)}) - f(x^*)) + \alpha_k^2\|g^{(k)}\|^2.$$

Ans: Let us consider,

$$\begin{aligned}\|x^{(k+1)} - x^*\|_2^2 &= \|x^{(k)} - \alpha_k g^{(k)} - x^*\|_2^2 \text{ [substituting for } x^{(k+1)}] \\ &= \|x^{(k)} - x^*\|_2^2 - 2\alpha_k g^{(k)T}(x^{(k)} - x^*) + \alpha_k^2\|g^{(k)}\|_2^2 \\ d_{k+1}^2 &= d_k^2 - 2\alpha_k g^{(k)T}(x^{(k)} - x^*) + \alpha_k^2\|g^{(k)}\|_2^2\end{aligned}$$

Now, from the definition of sub-gradients,

$$\begin{aligned}f(x^*) &\geq f(x^{(k)}) + g^{(k)T}(x^* - x^{(k)}) \\ f(x^*) - f(x^{(k)}) &\geq g^{(k)T}(x^* - x^{(k)})\end{aligned}$$

Therefore,

$$d_{k+1}^2 \leq d_k^2 - 2\alpha_k(f(x^{(k)}) - f(x^*)) + \alpha_k^2\|g^{(k)}\|_2^2 \tag{1}$$

(b) Using the above show that

$$2\sum_{j=1}^{k}\alpha_j(f(x^{(j)}) - f(x^*)) \leq R^2 + \sum_{j=1}^{k}\alpha_j^2\|g^{(j)}\|^2$$

Ans: Using equation (1) recursively we have,

$$d_{k+1}^2 \leq d_1^2 - 2\sum_{j=1}^{k}\alpha_j(f(x^j) - f(x^*)) + \sum_{j=1}^{k}\alpha_j^2\|g^{(j)}\|_2^2$$

Here, the first term is a telescopic sum. Hence only the first and the last term remains.
Now, Given $d_1 \leq R$

$$d_{k+1}^2 + 2\sum_{j=1}^{k} \alpha_j (f(x^j) - f(x^*)) \leq d_1^2 + \sum_{j=1}^{k} \alpha_j^2 \|g^{(j)}\|_2^2$$
$$d_{k+1}^2 + 2\sum_{j=1}^{k} \alpha_j (f(x^j) - f(x^*)) \leq R^2 + \sum_{j=1}^{k} \alpha_j^2 \|g^{(j)}\|_2^2$$

(c) Now show that,

$$f_{\text{best}}^{(k)} - f(x^*) \leq \frac{R^2 + G^2 \sum_{j=1}^{k} \alpha_j^2}{2\sum_{j=1}^{k} \alpha_j},$$

where $f_{\text{best}}^{(k)} = \min(f(x^{(1)}), \ldots, f(x^{(k)}))$.
Ans: Now,

$$\sum_{j=1}^{k} \alpha_j (f(x^j) - f(x^*)) \geq \sum_{j=1}^{k} \alpha_j \min(f(x^j) - f(x^*))$$

This is because subgradient need not decrease at each step. It is not a descent method.

$$\sum_{j=1}^{k} \alpha_j (f() - f(x^*)) \geq (\sum_{j=1}^{k} \alpha_j) \min((f(x^{(1)}) - f(x^*)), ..., (f(x^{(k)}) - f(x^*)))$$
$$\sum_{j=1}^{k} \alpha_j (f() - f(x^*)) \geq (\sum_{j=1}^{k} \alpha_j)(f_{best}^{(k)} - f(x^*))$$

Using the previous result

$$2(\sum_{j=1}^{k} \alpha_j)(f_{best}^{(k)} - f(x^*)) \leq R^2 + \sum_{j=1}^{k} \alpha_j^2 \|g^{(j)}\|_2^2$$
$$f_{\text{best}}^{(k)} - f(x^*) \leq \frac{R^2 + G^2 \sum_{j=1}^{k} \alpha_j^2}{2\sum_{j=1}^{k} \alpha_j} \text{ [given } \|g^{(j)}\|_2^2 \leq G]$$

(d) Find a sequence of α_i's such that the above is $O(\log k/\sqrt{k})$. Give a proof.
Ans: For the choice of $= 1/\sqrt{k}$ the required computation time is obtained.

$$2(\sum_{j=1}^{k} \alpha_j)(f_{best}^{(k)} - f(x^*)) \leq R^2 + \sum_{j=1}^{k} \alpha_j^2 \|g^{(j)}\|_2^2$$
$$f_{\text{best}}^{(k)} - f(x^*) \leq \frac{R^2 + G^2 \frac{1}{\sqrt{k}}}{2\frac{1}{\sqrt{k}}} \text{ [given } \|g^{(j)}\|_2^2 \leq G]$$

2. Consider a design matrix $\boldsymbol{X}$ such that $\boldsymbol{X}$ has orthonormal columns, i.e. $\boldsymbol{X^T X} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the $p \times p$ identity matrix. Consider the following regularization:

$$\min_{\beta} \frac{1}{2}(\boldsymbol{X\beta - y})^T(\boldsymbol{X\beta - y}) + \lambda||\boldsymbol{\beta}||_m \tag{2}$$

(a) Derive the solution to equation (1) for $m = 2$ (ridge regression).
Ans: let us consider the optimization function,

$$\begin{aligned}
&= \min_{\beta} \frac{1}{2}(\boldsymbol{X\beta - y})^T(\boldsymbol{X\beta - y}) + \lambda||\boldsymbol{\beta}||_2 \\
&= \min_{\beta} \frac{1}{2}(\boldsymbol{\beta^T X^T - y^T})(\boldsymbol{X\beta - y}) + \lambda(\boldsymbol{\beta^T \beta}) \\
&= \min_{\beta} \frac{1}{2}((\boldsymbol{y^T y + \beta^T X^T X\beta - \beta^T X^T y - y^T X\beta}) + \lambda(\boldsymbol{\beta^T \beta})
\end{aligned}$$

Taking the derivative w.r.t B we get,

$$\begin{aligned}
\nabla(L(\beta)) &= (\boldsymbol{X^T X\beta - X^T y + \lambda\beta}) \\
\hat{\boldsymbol{\beta}} &= (\boldsymbol{X^T X + \lambda I})^{-1}\boldsymbol{X^T y}
\end{aligned}$$

(b) Derive the solution to equation (1) for $m = 1$ (lasso).
Ans: let us consider the objective function,

$$\begin{aligned}
L(\beta) &= \min_{\beta} \frac{1}{2}(\boldsymbol{X\beta - y})^T(\boldsymbol{X\beta - y}) \\
&= \min_{\beta} \frac{1}{2}(\boldsymbol{\beta^T X^T - y^T})(\boldsymbol{X\beta - y}) \\
&= \min_{\beta} \frac{1}{2}((\boldsymbol{y^T y + \beta^T X^T X\beta - \beta^T X^T y - y^T X\beta}) \\
&= \min_{\beta} \frac{1}{2}(\boldsymbol{y^T y - y^T X X^T y + (\beta - X^T y)^T(\beta - X^T y)}) \text{ [Rearranging]} \\
&= \min_{\beta} \frac{1}{2}((\boldsymbol{\beta - X^T y})^{\boldsymbol{T}}(\boldsymbol{\beta - X^T y})
\end{aligned}$$

Taking the derivative w.r.t B we get,

$$\begin{aligned}
\nabla(L(\beta)) &= \frac{1}{2}(\boldsymbol{2\hat{\beta} - 2X^T y}) \\
\hat{\boldsymbol{\beta}} &= \boldsymbol{X^T y}
\end{aligned}$$

Now,

$$\tilde{\beta} = \operatorname*{argmin}_{\beta} \frac{1}{2}(\boldsymbol{\beta - \hat{\beta}})^T(\boldsymbol{\beta - \hat{\beta}}) + \lambda||\boldsymbol{\beta}||_1$$

For subgradients to exist

$$0 \in -\boldsymbol{X^T}(\boldsymbol{y} - \boldsymbol{X\beta}) + \boldsymbol{\lambda}(||\boldsymbol{\beta}||_{\mathbf{1}})$$

Thus, $\boldsymbol{X^T}(\boldsymbol{y} - \boldsymbol{X\beta}) = \boldsymbol{\lambda v}$ where,

$$v = \begin{cases} \{1\} & \text{if } \boldsymbol{\beta_i} \geq \mathbf{0} \\ \{-1\} & \text{if } \boldsymbol{\beta_i} \leq \mathbf{0} \\ [-1, 1] & \text{if } \boldsymbol{\beta_i} = \mathbf{0} \end{cases}$$

Thus,

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X^T y} - v\boldsymbol{I})$$

(c) When $m = 0$, the penalty involves $||\boldsymbol{\beta}||_0 = \sum_{i=1}^{p} \mathbb{1}(\beta_i \neq 0)$.

i. Is this a convex optimization problem? Why or why not?

Ans:Yes, this is a convex optimization problem because the loss fucntion is still of a convex nature. Only because of the regularizer term the computation of optimal β will require the usage of different cases.

ii. Show that the solution to equation (1) with $m = 0$ is given by $\tilde{\boldsymbol{\beta}}$, where

$$\tilde{\beta}_i = \begin{cases} \boldsymbol{v_i^T y}, & \text{if } |\boldsymbol{v_i^T y}| > \sqrt{2\lambda} \\ 0 & \text{if } |\boldsymbol{v_i^T y}| \leq \sqrt{2\lambda} \end{cases}$$

This is also called the hard thresholding estimator. $\boldsymbol{v_i}$ is the i^{th} column of $\boldsymbol{X}$.

Ans: let us consider the objective function,

$$\begin{aligned} L(\beta) &= \min_{\beta} \frac{1}{2}(\boldsymbol{X\beta} - \boldsymbol{y})^T(\boldsymbol{X\beta} - \boldsymbol{y}) \\ &= \min_{\beta} \frac{1}{2}(\boldsymbol{\beta^T X^T} - \boldsymbol{y^T})(\boldsymbol{X\beta} - \boldsymbol{y}) \\ &= \min_{\beta} \frac{1}{2}((\boldsymbol{y^T y} + \boldsymbol{\beta^T X^T X\beta} - \boldsymbol{\beta^T X^T y} - \boldsymbol{y^T X\beta}) \\ &= \min_{\beta} \frac{1}{2}(\boldsymbol{y^T y} - \boldsymbol{y^T X X^T y} + (\boldsymbol{\beta} - \boldsymbol{X^T y})^{\boldsymbol{T}}(\boldsymbol{\beta} - \boldsymbol{X^T y}) \text{ [Rearranging]} \\ &= \min_{\beta} \frac{1}{2}((\boldsymbol{\beta} - \boldsymbol{X^T y})^{\boldsymbol{T}}(\boldsymbol{\beta} - \boldsymbol{X^T y}) \end{aligned}$$

Taking the derivative w.r.t B we get,

$$\nabla(L(\beta)) = \frac{1}{2}(\mathbf{2}\hat{\boldsymbol{\beta}} - \mathbf{2}\boldsymbol{X^T y})$$

$$\hat{\boldsymbol{\beta}} = \boldsymbol{X^T y}$$

Now,

$$\tilde{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{2}(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}})^T(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}) + \lambda||\boldsymbol{\beta}||_0$$

Splitting up $||\boldsymbol{\beta}||_0$ we get,

$$\tilde{\beta}_i = \underset{\beta_i}{\operatorname{argmin}} \frac{1}{2}(\boldsymbol{\beta_i} - \hat{\boldsymbol{\beta}}_{\boldsymbol{i}})^2 + \lambda \sum_{i=1}^{p} \mathbb{1}(\beta_i \neq 0)$$

$$\tilde{\beta}_i = \underset{\beta_i}{\operatorname{argmin}} \frac{1}{2}(\boldsymbol{\beta_i} - \boldsymbol{v_i^T y})^2 + \lambda \sum_{i=1}^{p} \mathbb{1}(\beta_i \neq 0)$$

and this occurs if $\frac{1}{2}\boldsymbol{v_i^T y} > \lambda$
thus,

0 if $\text{—}\boldsymbol{v_i^T y}\text{—} \leq \sqrt{2\lambda}$