Support Vector Machines: Convex Optimization and Beyond

Hitarth Choubisa
Electrical & Computer Engineering Department
University of Toronto
Toronto, Canada
hitarth.choubisa@mail.utoronto.ca

***Abstract*—The present report discusses Support Vector Machines (SVM). I introduce the theory of SVM and then, discuss the limitations we face with the traditional approach, specifically computational costs and large data limitations. I then discuss the solutions to these challenges using quantum computing. I implement and test out one of the approaches and report the results. At the end, I discuss the limitations we face today and the follow-up work.**

I. Introduction

Support Vector Machines (SVM) were first introduced by Vapnik and his colleagues in 1992 [1] . SVMs are interesting machine learning models due to the fact that unlike deep learning models we can analyze them analytically and they can be used to learn very complex data using kernel trick [2] [3].

In this report, first I will investigate the origin of these two challenges for SVM by analyzing its formulation and the associated optimization problem. With the challenges being clear, I will then borrow ideas from Quantum computing to discuss and show how we can handle these challenges.

II. Classical SVM

A. Problem formulation

SVM classifier the points by creating a hyperplane that separates the two classes with maximum margin. Mathematically, this translates to maximizing $\frac{||g(x)||}{||w||}$ where $g(x) = w^T x + b$. We want to find w and b such that $g(x)$ is $+1$ for one class and -1 for the other class. This means that now the objective becomes to minimize,

$$J(w) = \frac{||w||^2}{2} \tag{1}$$

subjected to,

$$y_i(w^T x_i + b) \geq 1, i = 1, 2..., N$$

The underlying assumption here is that we can perfectly separate the two classes. However, this is not always the case. In that case, we work with soft-margin SVM to allow misclassification error. The problem gets modified to,

$$J(w) = \frac{||w||^2}{2} + C\sum_{i=1}^{N} \xi_i \tag{2}$$

subjected to,

$$y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, 2..., N$$

$$\xi_i \geq 0, i = 1, 2..., N$$

In dual form, the soft margin SVM formulation is,

$$max_\lambda(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j x_i x_j) \tag{3}$$

subjected to,

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

$$0 \leq \lambda_i \leq C, i = 1, 2..., N$$

If x_i's are not one-dimensional, the objective gets modified to,

$$max_\lambda(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^T x_j) \tag{4}$$

However, sometimes the problem is not linearly separable but separable in some other transformed space of the inputs. In such cases, we use kernel functions to transform the input data points to a higher dimensional space where the points may be linearly separable i.e. $x \rightarrow \phi(x)$ is the transformation. This transforms the formulation to,

$$max_\lambda(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j K(x_i, x_j)) \tag{5}$$

subjected to,

$$\begin{aligned} 0 \leq \lambda_i \leq C \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{6}$$

where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. In this case, the predicted output for a test point x_t is given by,

$$y_t = sign(\sum_{i=1}^{N} \lambda_i y_i K(x_i, x_t) + b) \tag{7}$$

where $b = y_k - \sum_i \lambda_i K(x_k, x_i)$ for a k such that $C > \alpha_k > 0$

B. Observations from the classical formulation

The problem defined in (5)-(6) is a quadratic program with N variables λ_i. We can make two quick observations from this formulation:

- We require $\mathcal{O}(N^2)$ evaluations to form the kernel matrix. Thus, the computational complexity for an entry of the kernel matrix scales as $\mathcal{O}(N^2 d)$ where d is some function of the input dimension M and N is the number of datapoints.
- The complexity associated with solving the quadratic program is $\mathcal{O}(N^3)$

Based on these observations, the overall complexity of the problem becomes $\mathcal{O}(N^2(N+d))$. When we have complex kernels, the evaluations are very expensive and it becomes to difficult to scale to larger datasets. Also, solving the quadratic program is cubic in the number of datapoints. Thus, along with the challenge of evaluation of the kernel matrix, we face the challenge of training the SVM. This makes us question if if it is possible to go beyond these restrictions and make this process more efficient. The answer is yes and it is entangled with the world of quantum computing.

III. Overcoming the challenges using Quantum Computing

In this section, I will discuss how we can use different quantum computing paradigms to accelerate and scale kernel based SVM. The discussion is inspired from the following 3 papers: [5] [6] [7]. [5] introduces mapping input space to a infinite dimensional hilbert space to enable efficient kernel evaluations using photonic quantum computers. [6] introduces an alternative strategy to use gate based quantum computers for efficient kernel evaluations. As part of the project, I tested and here demonstrate the performance quantum SVM using a Python [12] implementation of it. However, both these papers still rely on traditional optimization methods to train the SVM. To scale to larger datasets, we not only need efficient kernel evaluations but also faster training. Based on [7], I will finally discuss how we can accelerate this component using a specific formulation of SVM.

A. SVM in feature Hilbert space

In this section, I will discuss how we can accelerate one of the limiting components of SVM or kernel based methods in general and that's kernel evaluations. Discussions in this section is based on [5] and [6]. Recall, the primal form of SVM using (2). In a feature mapped representation, we can reformulate it as,

$$J(w) = \frac{||w||^2}{2} + C\sum_{i=1}^{N}\xi_i \tag{8}$$

subjected to,

$$\begin{aligned} y_i(w^T\phi(x_i)+b) \geq 1-\xi_i, i = 1, 2...N \\ \xi_i \geq 0, i = 1..., N \end{aligned} \tag{9}$$

From quantum computing perspectives, an analogue of the feature map $x \rightarrow \phi(x)$ corresponds to quantum feature map $x \rightarrow |\phi(x)\rangle$ which is obtained when a state preparation circuit $U_\phi(x)$ acts on the ground state $|0..0\rangle$ i.e. $|\phi(x)\rangle = U_\phi(x)|0..0\rangle$. Similar to the hyperplane in our primal SVM, we can a quantum circuit W such that $W|0..0\rangle = |w\rangle$. Depending on where $\phi(x)\rangle$ lies wrt to $w\rangle$, we have three possible scenarios:

- If $|\phi(x)\rangle$ is orthogonal to $|w\rangle$, then x lies on the decision boundary
- If $\langle w|\phi(x)\rangle < 0 (> 0)$, then x lies on the right (left) side of the hyperplane

Performing the gate operation U_ϕ, the process of encoding also performs the process of projecting to a higher dimensional space. Naturally, the way this is performed on a physical quantum computer will vary from one type of quantum computer to another. [5] showcases of how to do it using a photonic quantum computer whereas [6] shows how to do it on a universal gate based quantum computer.

By parameterizing the weights W, we train the quantum SVM i.e. we choose $|w\rangle = W(\theta)|0..0\rangle$ where θ is a parameter. Then using classical optimization techniques, we can train our quantum SVM (qSVM). We obtain a clear quantum advantage when we use kernel functions that are difficult to calculate on classical computers. One such kernel function introduced by [6] is the map given as,

$$\begin{aligned} U_\phi(x) = exp(i\sum_{S\subseteq[n]}\phi_S(x)\prod_{i\in S}Z_i) \\ \implies U_\phi(x) = exp(ix_1Z_1 + x_2Z_2 + (\pi - x_1)(\pi_{x2})Z_1Z_2) \end{aligned} \tag{10}$$

wherein the former expression simplifies to the later if we consider an input 2D vector $x = (x_1, x_2)$ and limit only at most 2 qubits interacting at a time with each other. In general, we can get a significant speed-up when we use quantum kernels. Getting a speed-up with classically easy-to-calculate kernels in general, doesn't give us any significant speedup.

There is a solution to exponentially accelerate the classical training process of the SVM. I will discuss it in section 2C.

B. Experiments with qSVM

Using the formulation from the previous section, I test it using IBM quantum computing library QISKIT [11]. For demonstration purposes, I used the breast cancer dataset which was PCAed to only 2 input dimensions. Using the qSVM, we obtain an accuracy of 92% which is slightly better than the traditional SVM (90%) indicating that quantum computing can potentially help accelerate SVM in a significant way in the future (Refer to Figure 1). The challenge is to design application specific appropriate quantum kernels!

C. SVM for big data classification

So far, we discussed potential way through which computational complexity of kernels can be brought down. However, scaling SVM with increasing number of datapoints is difficult

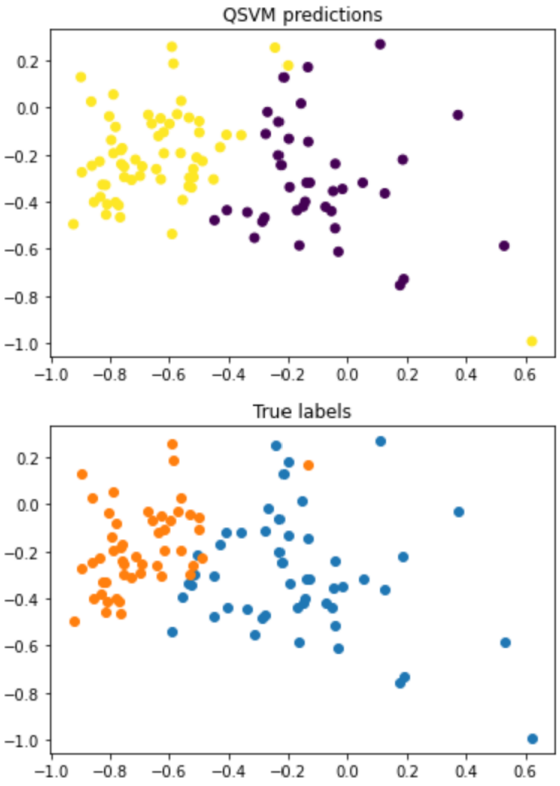

Fig. 1. Performance of qSVM

as well (refer to section 1B). [7] introduced how we can accelerate least squared SVM using quantum computing. I will adapt some of their findings for this section. Then towards the end, talk about the future perspectives and limitations.

Recall, from (8) that we have the second term that roughly corresponds to L1 error term. In 1999, [8] introduced a least squares based SVM formulation wherein they replaced the linearly error term with least squared error term. This modifies the formulation of (8) as,

$$J(w,\xi) = \frac{||w||^2}{2} + \frac{\gamma}{2}\sum_{i=1}^{N}\xi_i^2 \tag{11}$$

subjected to,

$$y_i(w^T\phi(x_i)+b) = 1-\xi_i, i = 1,2...N \tag{12}$$

We can define the lagrangian with lagrangian variables λ_k's as,

$$\mathcal{L}(w,b,\xi,\lambda) = J(w,\xi) - \sum_{k=1}^{N}\lambda_k\{y_k[w^T\phi(x_k)+b]-1+e_k\} \tag{13}$$

Using KKT conditiions for optimality can be written in the matrix form as,

$$F\begin{pmatrix} b \\ \vec{\alpha}\end{pmatrix} = \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & K+\gamma^{-1}\mathbb{1}\end{pmatrix}\begin{pmatrix} b \\ \vec{\alpha}\end{pmatrix} = \begin{pmatrix} 0 \\ \vec{y}\end{pmatrix} \tag{14}$$

where $K_{ij} = x_i^T x_j$, $\vec{y} = [y_1, y_2, .., y_M]$ and $\vec{1} = [1,1,...,1]^T$. The matrix F is $(N+1)\times(N+1)$ dimensional. The classical complexity associated with this quadratic program is $\mathcal{O}(N^3)$. We can develop an analogue of (14) in quantum information wherein the quantum state $|b,\vec{\alpha}\rangle$ describes the hyperplane and we can find it using an efficient matrix inversion enabled by quantum computer (similar to [9]). To predict the labels, we then perform swap test between $|b,\vec{\alpha}\rangle$ and $|x\rangle$. The procedure by 1001[7] is summarized below:

- We normalize the equation by $tr(F)$ i.e. we solve the normalized equation $\hat{F}|b,\vec{\alpha}\rangle = |\vec{y}\rangle$ where $\hat{F} = F/tr(F)$
- We can expand $\hat{F}$ as $\hat{F} = (J+K+\gamma^{-1}\mathbb{1})/tr(F)$ with

$$J = \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & 0\end{pmatrix} \tag{15}$$

- We can efficiently calculate matrix inversion using quantum computers if we can efficiently exponentiate it. Here we have,

$$e^{-i\hat{F}\Delta t} = e^{-i\Delta t\mathbb{1}/tr(F)}\cdot e^{-iJ\Delta t/tr(F)}\cdot e^{-iK\Delta t/tr(F)} \tag{16}$$

  Eigenstates corresponding to the two non-zero eigenvalues of J are

$$|\lambda_{\pm}^{star}\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm \frac{1}{\sqrt{N}}\sum_{k=1}^{N}|k\rangle)$$

  Thus calculating its exponential is easy and so will be its inverse using quantum computers. Similary, the matrix $\gamma^{-1}\mathbb{1}$ is trivial. For $K/tr(F)$, efficient matrix exponentiation is enabled by the strategy described in [10].
- As the next step, we invert the eigenvalue followed by controlled rotation as demonstrated in [9]. This yields the inverse of the matrix we desire and hence, the state $|b,\vec{\alpha}\rangle$ that defines our SVM

D. Observations

The above described algorithm reduces the time complexity of the SVM to $\mathcal{O}(log(NM))$ where N is the dataset size and M is the dimension size. This is exponential speedup as compared to the classical implementation that is $\mathcal{O}(M^2(M+N))$

IV. Conclusion

In this paper, first I summarize the traditional development of Support Vector Machine(SVM); a popular Machine learning technique. Then I discuss 2 ways through which we can accelerate the training of the optimization problems associated with SVM. I also demonstrate the performance of the first approach on a quantum computer. The second approach that provides exponential speedup was treated theoretically. Specifically because, it requires access to QRAM (quantum datasets) and less noisy quantum computers than those that are available publicly. Immediate follow-up work on this will include development of novel quantum kernels that can solve interesting problems.

REFERENCES

[1] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "Training algorithm for optimal margin classifiers," in Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, 1992, pp. 144–152

[2] M. Awad, R. Khanna, M. Awad, and R. Khanna, "Support Vector Regression," in Efficient Learning Machines, Apress, 2015, pp. 67–80.

[3] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn, "Using NLP techniques for file fragment classification," in Proceedings of the Digital Forensic Research Conference, DFRWS 2012 USA, Aug. 2012, vol. 9, pp. S44–S49

[4] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," Neurocomputing, vol. 408, pp. 189–215, Sep. 2020, doi: 10.1016/j.neucom.2019.10.118.

[5] M. Schuld and N. Killoran, "Quantum Machine Learning in Feature Hilbert Spaces," Phys. Rev. Lett., vol. 122, no. 4, p. 040504, Feb. 2019.

[6] V. Havlíček et al., "Supervised learning with quantum-enhanced feature spaces," Nature, vol. 567, no. 7747, pp. 209–212, 2019.

[7] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," Phys. Rev. Lett., vol. 113, no. 3, p. 130503, Sep. 2014.

[8] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," Neural Process. Lett., vol. 9, no. 3, pp. 293–300, 1999

[9] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," Phys. Rev. Lett., vol. 103, no. 15, p. 150502, Oct. 2009.

[10] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum principal component analysis," Nat. Phys., vol. 10, no. 9, pp. 631–633, Jul. 2014.

[11] H. Abraham et al., "Qiskit: An Open-source Framework for Quantum Computing." 2019.

[12] G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.