Project Final Report

Hong-Min Chu, Dan Hofman, Guanwen Yan

December 2019

1 Introduction

For this project, we present a survey of the current research on using quantum algorithms in the implementation of a Support Vector Machine (SVM). An SVM is a supervised learning model that classifies data into one of the two given categories. In particular, an SVM provides a decision boundary with margin to each category of data as large as possible. Implementing SVMs using quantum algorithms can result in significant speedup over the classical implementations, potentially bringing the originally polynomial complexity down to logarithmic scale. To understand how this is the case, we conduct preliminary review of related literature and provide an in-depth discussion of how exactly these quantum SVMs work in the following sections.

The formulation of SVM was first proposed by C. Cortes and V. Vapnik in [1]. It was shown that the optimal decision hyper-plane is the linear combination of a subset of data vectors, and hence comes the name "Support Vector Machine". Learning non-linear SVM can be reduced to solving convex quadratic optimization problem, which in the worst case takes time cubic to the number of data [2]. Previous studies [5, 2, 3] improve the learning algorithm to take time only quadratic to the number of data. They achieved so by exploiting special structure of SVMs formulation to solve the quadratic problem more efficiently. Another line of effort was dedicated to the linear SVM [6] which was popular in text data mining [7]. Time complexity of learning linear SVM is only linear to the number of data at the cost of less expressive power.

Comparing to classical sampling algorithms which require polynomial time complexity, quantum SVM has the complexity logarithmic in the size of the vectors and the number of training examples. Rebentrost et al. showed how support vector machines can be implemented on a quantum computer [8]. Before that paper, Lloyd et al. developed a fast quantum evaluation of inner products, which is useful in preparing the kernel matrix of SVMs [9]. Anguita et al. showed how to train SVM effectively using Quantum Computing [10]. They analyzed the behavioral of conventional and enhanced SVMs, and compared their theoretic results with experiments. Harrow, Hassidim and Lloyd showed how to use quantum algorithms to estimate solutions to linear systems of equations [11]. This algorithm is essential in training SVMs. Zhaokai Li et al. constructed a scheme to build quantum SVMs experimentally [12].

The rest of this survey is organized as follows: We review SVM in classical setting in Section 2. We then present the setup of quantum SVM in Section 3, and discuss its time complexity advantage in Section 4.

2 Classical Support Vector Machine

In this section, we give an in-depth review of the classical SVM. We start by discussing the origin of SVM families, hard-margin SVM, in Section 2.1. Then in Section 2.2, we describe improvement of hard-margin SVM based on non-linear transformation using the Lagrange dual, and discuss how kernel trick can be employed for further efficiency improvement. Next, we discuss soft-margin SVM, which improves upon hard-margin SVM to handle data that is not linear-separable, in Section 2.3. We then review least-square SVM in Section 2.4. Least-square SVM is another variant of SVM family whose computational efficiency can greatly benefit from quantum computing. Finally, we briefly summarize and compared the time complexity of different SVM variants in Section 2.5.

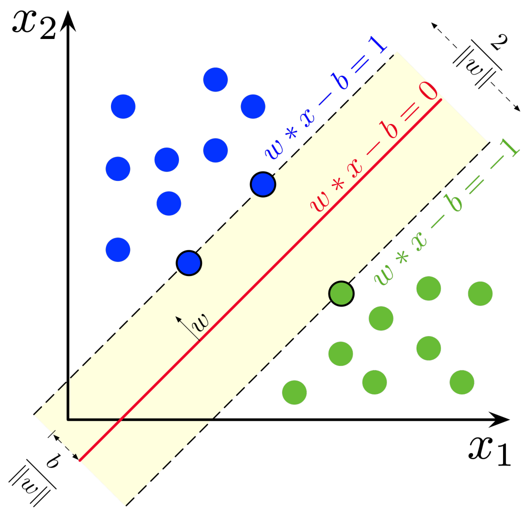

Figure 1: Illustration of hard-margin SVM

2.1 Hard-margin SVM

The input of classical SVM is a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^M$ with M instances, where each $\mathbf{x} \in \mathbb{R}^N$ and $y \in \{+1, -1\}$. Given the input dataset, a hard-margin SVM [1] aims to find a hyperp-plane $(\mathbf{w}, b)$ such that $\mathbf{w}^\top \mathbf{x} + b$ classifies each instance correctly. In addition, distance between $(\mathbf{w}, b)$ to the closest $\mathbf{x}$, or the margin, must be maximized. Fig. 1 illustrates the idea of hard-margin SVM in two-dimensional. In particular, the decision hyperplance is coloered in red, and the margin, which is cloroed in yellow, is determined by the closest distance between each group of data to the hyperplane. The above can be formulated as

$$\begin{aligned} \arg\min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} \\ \text{s.t.} \quad & y_i(\mathbf{w}_i^\top \mathbf{x}_i + b_i) \geq 1, \quad \forall i = 1, \ldots, M \end{aligned}$$

which is a standard quadratic programming (QP) problem.

2.2 Non-linear SVM and Kernel Trick

The first draw back of the formulation of hard-margin SVM is that classifier is linear, and practical applications often involve data exhibiting non-linear separation. One way to tackle such problem is to introduce $\phi(\cdot) : \mathbb{R}^N \times R^{\tilde{N}}$ which applies non-linear transformation to input feature $\mathbf{x}_i$ as $\mathbf{z}_i = \phi(\mathbf{x}_i)$ by mapping it to a higher-dimensional space of $\mathbb{R}^{\tilde{N}}$. Nevertheless, usage of $\phi(\mathbf{x})$ introduces $O(\tilde{M})$ variables in the QP formulation of hard-margin SVM. To avoid such dependency, one can introduce Lagrange multipliers $\{\alpha_i\}_{i=1}^M$ and rewrite the formulation as its dual form as

$$\begin{aligned} \arg\min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j \mathbf{z}_i^\top \mathbf{z}_j + \sum_{i=1}^{M} \alpha_i \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad \forall i = 1, \ldots, M \\ & \sum_{i=1}^{M} y_i \alpha_i = 0 \end{aligned}$$

The new QP problem now only has M variables and $M + 1$ constraints. One can recover $\mathbf{w}$ by $\mathbf{w} = \sum_{i=1}^M \alpha_i y_i \mathbf{z}_i$ and recover b as $\mathbf{b} = y_m - \mathbf{w}^\top \mathbf{z}_m$ by choosing any m where $\alpha_m > 0$.

One may observe that, while the QP problem itself is free from $O(\tilde{N})$ dependency, one still needs to compute $\mathbf{z}_i^\top \mathbf{z}_j$ a priori. Fortunately, we can speed up the process of feature transformation plus inner product by replacing it with a kernel function $k(\cdot, \cdot)$, where $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$. With carefully designed kernel, one can compute $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$ with time complexity $O(N)$ instead of $O(\tilde{N})$ for a wide range of ϕ. Incorporating kernel k into the (hard-margin) SVM is straight-forward by rewriting the above dual form as

$$
\begin{aligned}
\arg\min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i^\top \mathbf{x}_j) + \sum_{i=1}^{M} \alpha_i \\
\text{s.t.} \quad & \alpha_i \geq 0, \quad \forall i = 1, \dots, M \\
& \sum_{i=1}^{M} y_i \alpha_i = 0
\end{aligned}
$$

If kernel k satisfies Mercer's condition, the above optimization is a convex QP. Noticeably, in the kernel formulation, only b but not $\mathbf{w}$ can be obtained explicitly using complementary slackness. To perform prediction on a new x, one need to instead calculate $\sum_{m:\alpha_m \neq 0} \alpha_m y_m k(\mathbf{x}_n, \mathbf{x}) + b$.

2.3 Soft-margin SVM

Another drawback of hard-margin SVM is that to have feasible solutions, the data must be linear-separable (after applying ϕ). This clearly hinders the practical applications of hard-margin SVM. Therefore, soft-margin SVM, which is the commonly-seen form of SVM nowadays, was proposed. Unlike hard-margin SVM, soft-margin SVM allows the data to be classified incorrectly, and instead tries to balance between the margin and the distance from $(\mathbf{w}, b)$ to those wrongly-classified instances. The above intuition can be formulated as

$$
\begin{aligned}
\arg\min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{i=1}^{M} \xi_i \\
\text{s.t.} \quad & y_i(\mathbf{w}_i^\top \mathbf{x}_i + b_i) \geq 1 - \xi_i, \quad && \forall i = 1, \dots, M \\
& \xi_i \geq 0, && \forall i = 1, \dots, M
\end{aligned}
$$

where ξ_i here represents the 'violation' if x_i is at the wrong side of the margin (i.e. $y_i(\mathbf{w}_i^\top \mathbf{x}_i + b_i) < 1$) and C controls the trade-off between larger margin and smaller violation. The dual formulation of soft-margin SVM can be derived similarly as hard-margin SVM by

$$
\begin{aligned}
\arg\min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i \mathbf{x}_j) + \sum_{i=1}^{M} \alpha_i \\
\text{s.t.} \quad & \alpha_i \geq 0, \quad \forall i = 1, \dots, M \\
& \alpha_i \leq C, \quad \forall i = 1, \dots, M \\
& \sum_{i=1}^{M} y_i \alpha_i = 0
\end{aligned}
$$

The above QP is of N variables and $2N + 1$ constraints.

2.4 Least-square SVM

Least-square SVM (LS-SVM) is another variant of soft-margin SVM which employs square penalty to the violation of instance to the margin. Such alternative results in the following new formulation:

$$
\begin{aligned}
\arg\min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \gamma\sum_{m=1}^{M} \xi_m^2 \\
\text{s.t.} \quad & y_m(\mathbf{w}^\top \phi(\mathbf{x}_m) + b) = 1 - \xi_m, \quad \forall m = 1, \dots, M
\end{aligned}
$$

Applying the Lagrange dual, the optimal solution for LS-SVM is given by

$$\begin{bmatrix} 0 & \mathbf{1}^\top \\ \mathbf{1} & \mathbf{K} + \gamma^{-1}\mathbf{I} \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix}$$

where $\mathbf{K}(i,j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ and $\mathbf{w}$ is resembled by $\sum_{m=1}^{M} \alpha_m \phi(\mathbf{x}_m)$. When kernel trick is employed, we have $\mathbf{K}(i,j) = k(\mathbf{x}_i, \mathbf{x}_j)$ and the prediction is made by $\sum_{m=1}^{M} \alpha_m K(\mathbf{x}_m, \mathbf{x}) + b$.

The key difference between LS-SVM and other variants introduced above lies between the complexity of the induced optimization problem. In particular, we are only required to solve a linear system in LS-SVM, while we need to solve a convex QP for other variants of SVM. On the other hand, solving linear system in quantum computing is known to be efficient [11], which hints that the training efficiency of LS-SVM can greatly benefit from quantum computing, as we will discuss further in Section 3 and Section 4.

2.5 Time Complexity

Hard-margin SVM and soft-margin SVM which employs linear penalty on the violation requires us to solve a convex QP with inequality constraints. Optimization algorithms for general convex QP with inequality constraints involve an iterative procedure that requires inversion of the kernel matrix, whose time complexity is $O(N^3)$. Dedicated SVM training algorithms [2] exploits special structure of the induced QP to avoid matrix multiplication. Time complexity of such algorithms depend on the number of support vectors, and empirically scale between $O(N)$ to $O(N^2)$.

On the other hand, to obtain optimal solution for LS-SVM, we are required to solve a linear system. A direct matrix inversion has time complexity $O(N^3)$. It is also possible to rewrite the linear system to have positive-definite constraint matrix, where iterative methods like conjugate gradient can be applied [22]. The time complexity of iterative methods generally depend on the condition number of the constraint matrix, but the efficiency is better than matrix inversion in most practical applications.

3 Quantum Support Vector Machines

3.1 Setup

The input of a quantum support vector machine are M known quantum states

$$|\vec{x}_j\rangle = \frac{1}{|\vec{x}_j|} \sum_{k=1}^{N} (\vec{x}_j)_k \, |k\rangle \,, \tag{1}$$

their norms $|\vec{x}_j|$, and their labels $y_j \in \{-1, 1\}$ for each $j \in \{1, \cdots, M\}$.

We will assume that we have access to an oracle that generates these input states. However, we will briefly discuss one method of storing and generating these states using quantum RAM (qRAM). The use of qRAM allows us to access stored data in parallel.

For $\vec{v} \in \mathbb{C}^N$ with components $v_i = |v_i| \, e^{i\varphi_i}$, assume we have $\{|v_i|, \varphi_i\}_{i=1}^{N}$ stored in qRAM. Then we can construct state $|v\rangle = \frac{1}{|v|}\vec{v}$ in $O(\log N)$ steps as long as we are able to estimate sub-norms $n_\ell = \sum_{i=1}^{\ell} |v_i|^2$ efficiently [9]. With this setup, if we have M training examples, we can reconstruct their quantum states in time $O(\log MN)$.

3.2 Training the Model

The training of a SVM is to construct the kernel matrix K, and maximizing over the Karush-Kuhn-Tucker multipliers $\vec{\alpha} = (\alpha_1, \cdots, \alpha_M)^T$ the function:

$$L(\vec{\alpha}) = \sum_{j=1}^{M} y_j \alpha_j - \frac{1}{2} \sum_{j,k=1}^{M} \alpha_j K_{jk} \alpha_k, \tag{2}$$

subject to the constrains $\sum_{j=1}^{M} \alpha_j = 0$ and $y_j \alpha_j \geq 0$. To prepare the normalized kernel function $\hat{K} = K/Tr[K]$, We first prepare the state $|\chi\rangle = 1/\sqrt{N_\chi} \sum_{i=1}^{M} |\vec{x}_i| \, |i\rangle \, |\vec{x}_i\rangle$, where $N_\chi = \sum_{i=1}^{M} |\vec{x}_i|^2$. To prepare $\hat{K}$, we discard the training set register, or equivalently take the partial trace

$$Tr_2\{|\chi\rangle \langle\chi|\} = \frac{1}{N_\chi} \sum_{i,j=1}^{M} \langle \vec{x}_j | \vec{x}_i \rangle |\vec{x}_i| |\vec{x}_j| \, |i\rangle \langle j| = \frac{K}{Tr[K]} = \hat{K}. \tag{3}$$

Once we prepared the kernel matrix K, we can employ the least-square reformulation of the support vector machine and obtain the parameters $\vec{\alpha}$ from the solution of a linear equation system. Following the procedures in [14], we get a least-squares approximation of the problem:

$$F \begin{pmatrix} b \\ \vec{\alpha} \end{pmatrix} \equiv \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & K + \gamma^{-1}\mathbb{1} \end{pmatrix} \begin{pmatrix} b \\ \vec{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ \vec{y} \end{pmatrix} \tag{4}$$

where user-specified γ determines the relative weight of training error and SVM objective. For the training of a quantum SVM, our goal is to produce a quantum state $|b, \vec{\alpha}\rangle$. In order to do so, we need to solve the normalized $\hat{F}\,|b, \vec{\alpha}\rangle = |\vec{y}\rangle$, where $\hat{F} = F/Tr[F]$. To find the inverse of $\hat{F}$, we need to separate $\hat{F}$ as $\hat{F} = (J + K + \gamma^{-1}\mathbb{1})/Tr[F]$, with $J = \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & 0 \end{pmatrix}$. The generator of $\hat{F}$ can then be decomposed as

$$e^{-i\hat{F}\Delta t} = e^{-i\Delta t \mathbb{1}/Tr[F]} e^{-iJ\Delta t/Tr[F]} e^{-iK\Delta t/Tr[F]} + O(\Delta t^2). \tag{5}$$

The exponentiation of the matrix $\gamma^{-1}\mathbb{1}$ is trivial. The two eigenvalues of the matrix J are $\lambda_\pm = \pm\sqrt{M}$ and the corresponding eigenstates are $|\lambda_\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm \frac{1}{\sqrt{M}} \sum_{k=1}^{M} |k\rangle)$. We can use these information to calculate $e^{-iJ\Delta t/Tr[F]}$. For $\hat{K}$, we apply the algorithm in [15]:

$$\begin{aligned} e^{-i\hat{K}\Delta t} \rho e^{i\hat{K}\Delta t} &= e^{-iL_{\hat{K}}\Delta t}(\rho) \\ &\approx \rho - i\Delta t[\hat{K}, \rho] + O(\Delta t^2) \end{aligned} \tag{6}$$

We can use this to get the inverse of $\hat{K}$. For the vector $|\vec{y}\rangle$, we can expand it into eigenstates $|u_j\rangle$ of $\hat{F}$ with corresponding eigenvalues λ_j, $|\tilde{y}\rangle = \sum_{j=1}^{M+1} \langle u_j | \tilde{y} \rangle \, |u_j\rangle$. Then we follow the steps in [11], performing a controlled rotation and uncomputing the eigenvalue register. Thefinal stste we have is $\sum_{j=1}^{M+1} \frac{\langle u_j | \tilde{y}}{\lambda_j} |u_j\rangle$. In the basis of training set labels, the expansion coefficients of this state are the desired SVM parameters:

$$|b, \vec{\alpha}\rangle = \frac{1}{\sqrt{C}} \left(b\,|0\rangle + \sum_{k=1}^{M} \alpha_k \,|k\rangle \right) \tag{7}$$

where $C = b^2 + \sum_{k=1}^{M} \alpha_k \,|k\rangle$.

3.3 Classification Using the Model

After finishing training the SVM, we can now classify a query state $|\vec{x}\rangle$. Using the parameters we got in equation 7, we can construct the state

$$|\tilde{u}\rangle = \frac{1}{\sqrt{N_{\tilde{u}}}} \left(b\,|0\rangle\,|0\rangle + \sum_{k=1}^{M} \alpha_k |\vec{x}_k| \, |k\rangle \, |\vec{x}_k\rangle \right), \tag{8}$$

with $N_{\tilde{u}} = b^2 + \sum_{k=1}^{M} \alpha_k^2 |\vec{x}_k|^2$. Next, we construct the query state:

$$|\tilde{x}\rangle = \frac{1}{\sqrt{N_{\tilde{x}}}} \left(|0\rangle\,|0\rangle + \sum_{k=1}^{M} |\vec{x}| \, |k\rangle \, |\vec{x}\rangle \right) \tag{9}$$

with $N_{\tilde{x}} = M|\vec{x}|^2 + 1$. For the classification, we perform a swap test. We construct the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle |\tilde{u}\rangle + |1\rangle |\tilde{x}\rangle)$ and measure it in the state $|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The measuement has the success probability $P = |\langle\psi|\phi\rangle|^2 = \frac{1}{2}(1 - \langle\tilde{u}|\tilde{x}\rangle)$. The inner product is given by

$$\langle\tilde{u}|\tilde{x}\rangle = \frac{1}{\sqrt{N_{\tilde{x}} N_{\tilde{u}}}}(b + \sum_{k=1}^{M} \alpha_k |\vec{x}_k||\vec{x}|\langle\vec{x}_k|\vec{x}\rangle).$$

If $P < \frac{1}{2}$ we classify $|\vec{x}\rangle$ as +1, otherwise -1.

3.4 Kernel Matrix Approximation and Error Analysis

We now show that quantum matrix inversion essentially performs a kernel matrix principal component analysis and give an error analysis of the quantum algorithm. The matrix under consideration, $\hat{F} = F/\operatorname{tr} F$, contains the kernel matrix $\hat{K} = K/\operatorname{tr} K$ and an additional row and column due to the offset parameter b. In case the offset is negligible, the problem reduces to matrix inversion of the kernel matrix $\hat{K}_\gamma$ only. For any finite γ, $\hat{K}_\gamma$ is positive definite, and thus inevitable. The positive eigenvalues of $\hat{F}$ are dominated by the eigenvalues of $\hat{K}_\gamma$. In addition, $\hat{F}$ has one additional negative eigenvalue which is involved in determining the offset parameter b. The maximum absolute eigenvalue of $\hat{F}$ is no greater than 1 and the minimum absolute eigenvalue is $\leq O(1/M)$. The minimum eigenvalue arises e.g. from the possibility of having a training example that has very little (almost zero) overlap with the other training examples. Because of the normalization the eigenvalue will be $O(1/M)$ and as a result the condition number κ (largest eigenvalue divided by smallest eigenvalue) is $O(M)$ in this case. To resolve such an eigenvalue would require exponential runtime. We define a constant ϵ_K such that only the eigenvalues in the interval $\epsilon_K \leq |\lambda_j| \leq 1$ are taken into account, essentially defining an effective condition number $\kappa_{eff} = 1/\epsilon_K$. Then, the filtering procedure is employed in the phase estimation using this κ_{eff}. An ancilla register is attached to the quantum state and appropriately defined filtering functions discard eigenvalues below ϵ_K when multiplying the inverse $1/\lambda_j$ for each eigenstate. The desired outcome is obtained by post-selecting the ancilla register.

The legitimacy of this eigenvalue filtering can be rationalized by a principal component analysis (PCA) argument. Define the $N \times M$ (standardized) data matrix $X = (\vec{x}_1, \cdots \vec{x}_M)$. The $M \times M$ kernel matrix is given by $K = X^T X$. The $N \times N$ covariance matrix is given by $\Sigma = XX^T = \sum_{m=1}^{M} \vec{x}_m \vec{x}_m^T$. Often data sets are effectively described by a few unknown factors (principal components), which admit a low-rank approximation for Σ. This amounts to finding the eigenvectors $\vec{v}_i$ of Σ with the largest eigenvalues λ_i. The matrices XX^T and $X^T X$ have the same non-zero eigenvalues. Keeping the large eigenvalues and corresponding eigenvectors of the kernel matrix thus retains the principal components of the covariance matrix, i.e. the most important features of the data. Our quantum speed-up holds in the case when the data is in a low-rank situation, i.e. the kernel matrix has a few $O(1)$ eigenvalues and many $O(1/M)$ eigenvalues, all of them initially unknown. The quantum algorithm only takes into account the $O(1)$ eigenvalues, which incurres an error E. This error is given by the norm of the difference between the low-rank matrix K and its quantum approximation K_q, i.e. $E = ||K - K_q||_F$, using the Frobenius/Hilbert-Schmidt norm. This error is given by $E = \sqrt{\sum_{\lambda_i = O(1/M)} \lambda_i^2}$, where λ_i are the eigenvalues of K. Since by assumption we have $O(M)$ small eigenvalues, this error is $E = O(1/\sqrt{M})$.

Regarding the cutoff ϵ_K, $O(1)$ eigenvalues of $K/\operatorname{tr} K$ exist for example in the case of well-separated clusters with $O(M)$ vectors in them. A simple artificial example is $K = \mathbb{1}_{2\times 2} \bigotimes (\vec{1}\vec{1}^T)_{M/2 \times M/2}$. Note that finding the principal components of the kernel matrix is performed in quantum parallel by phase estimation and the filtering procedure.

4 Time Complexity of Quantum SVMs

In this section we will compare the time complexity of classical SVMs to quantum SVMs. Because the quantum SVM we discuss is a least-squares model, we will compare it to a least-squares classical SVM. We first discuss the time complexity of training the model and after we discuss the time complexity of classifying an unseen example using the model.

4.1 Training the Model

We first begin with a discussion of the algorithms to train or prepare the SVM model. The two central steps to the least-squares SVM algorithms outlined above are computing the kernel matrix and solving the least-squares problem. The quantum algorithm yields improvements over the classical algorithm in both of these parts, which explains its logarithmic running time.

In classical SVM, each entry of the kernel matrix takes time $O(N)$ to compute. This is the time that it takes to calculate the dot product between two N-dimensional vectors. There are also $O(M^2)$ entries in the kernel matrix since we need an entry for each pair of training vectors. Hence, finding the kernel matrix takes time $O(M^2N)$.

Also in classical SVM, solving the least-squares problem takes $O(M^3)$ time, which is the time it takes to solve the QP problem. Therefore, the total time for the classical SVM algorithm is $O\big(M^2(N+M)\big)$.

Now we review these two steps for the quantum SVM algorithm. Our first major improvement is in calculating the kernel matrix $\hat{K} = K/\text{tr}(K)$ where $K_{ij} = \langle \vec{x}_i | \vec{x}_j \rangle$.

Using the algorithm in [9], we can approximate the inner product between two N-dimensional ($\log_2 N$-qubit) states in time $O(\epsilon^{-1}\log N)$ with accuracy ϵ. This is done by estimating $|x_i|^2 + |x_j|^2$ and $|x_i - x_j|^2$ and then computing $x_i^T x_j = \frac{1}{2}\left(|x_i|^2 + |x_j|^2 - |x_i - x_j|^2\right)$. However, to compute the entire kernel matrix, we still need to evaluate $\binom{M}{2} = O(M^2)$ inner products, which would yield K in time $O(M^2 \log N)$ using quantum inner product.

We can more efficiently prepare the entire kernel matrix at once by using a method we reviewed in the section above. To prepare $\hat{K}$ we call the training data oracles on the state $\frac{1}{\sqrt{M}}\sum_{i=1}^{M}|i\rangle$, which is how we obtain the state $|\chi\rangle$ with partial trace $\hat{K}$. Because using quantum RAM allows for $O(\log MN)$ access time, we have that the preparation of $\hat{K}$ takes time $O(\log MN)$.

Finally, once we have $\hat{K}$, we only need to compute $\text{tr}(K)$ in order to recover K. We can approximate the trace of the kernel matrix efficiently using the method in [8] by generating the Hamiltonian using the norms of the training data stored in the quantum RAM.

Now we review how to solve the least-squares problem quickly using quantum operations. Instead of solving the QP as in the classical algorithm, we evaluate the inverse of F in Equation 4 to find a solution. Using the Lie Product, we find that we can estimate the inverse of F by simulating $e^{-i\gamma^{-1}\mathbb{1}\Delta t}e^{-iJ\Delta t}e^{-iK\Delta t}$. J and $\gamma^{-1}\mathbb{1}$ are sparse, so we can easily simulate them using the method in [16]. K, however, is not sparse, so we must use a separate technique, called quantum self-analysis to simulate $e^{i\hat{K}\Delta t}$ in time $O(\log MN)$ [17]. Quantum self-analysis involves creating multiple copies of the density matrix $\hat{K}$ to aid in the simulation. Using these methods together, we obtain a total time of $O(\log MN)$ to solve the least-squares problem.

A careful analysis of these techniques and the errors they accrue in [8] reveals that we can approximate a solution in time $O(\epsilon^{-3}\epsilon_K^{-3}\log MN)$ time, where we can choose ϵ and ϵ_K lower to make the approximation more accurate. We choose ϵ to control the number of iterations in the algorithm. We choose ϵ_K so that during the inversion of $\hat{K}$, only eigenvalues with $\epsilon_K \leq |\lambda_i| \leq 1$ are considered. This creates an effective condition number of $\frac{1}{\epsilon_K}$ so that the inversion can be computed efficiently. Hence, we have that the classical algorithm takes time $O\big(M^2(M+N)\big)$ to prepare the model, while the quantum algorithm takes time $O(\log MN)$. We should keep in mind though that this is limited only to least-squares SVM, which is not so commonly used. Also, the quantum algorithm is an approximation algorithm with approximation factors suppressed in the running time, while the classical algorithm is not.

4.2 Classification Using the Model

We now compare the running time of prediction using our SVM models in the classical case and quantum case.

Prediction using the classical model takes $O(M)$ times because once we have a solution, we can produce a prediction by direct calculation using the M values for α.

In the quantum model, classification with accuracy ϵ requires iterating $O(P(1-P)/\epsilon^2)$ times, where $P = \frac{1}{2}(1 - \langle \tilde{u} \,|\tilde{x}\rangle)$ [8].

These two running times are quite different. In general, classification in the quantum SVM is faster when M is large, but classical SVM may be preferable if accuracy is important.

5 Limitations and Open Problems

In this section, we discuss some of the limitations of the quantum SVM model proposed when compared to classical SVM models and then we discuss some open problems about quantum SVMs that are currently being studied.

5.1 Limitations

Even though the running time of the least-squares running time is much better for the quantum version than the classical version, there are many limitations to the quantum SVM model we have outlined. Because of these limitations, the quantum model is not always an obvious better choice.

As discussed before, this quantum algorithm only works for least-squares SVM, but this is a less commonly used version of SVM because it can lead to more overfitting. It is more common to use linear penalty of violation rather than squared penalty in the classical SVM.

In addition, the method of calculating the kernel matrix outlined above only works for polynomial kernels. This method will not work for RBF kernels for example, which are very commonly used in classical SVM. Furthermore, the kernel matrix inversion algorithm discussed above requires that K is Hermitian, so the kernels must also be symmetrix for quantum SVM.

Another important point to note is that the running time $O\big(M^2(M+N)\big)$ for classical least-squares SVM is only a worst-case running time that is only realized if every entry of the kernel matrix must be calculated and the QP problem is sufficiently hard. This running time is reasonable for dense training vectors, but if the training vectors are sparse, calculating the kernel matrix and solving the least-squares problem can be much faster since most operations do not need to be done. There are many classification problems that employ sparse training vectors in which using the quantum SVM might not actually give a benefit over classical SVM.

Because of these restrictions, using quantum SVM might only be beneficial in classification problems where the training vectors are dense and polynomial kernals are useful.

5.2 Open Problems

There are many open problems in quantum SVM to be left to explore. This paper only discusses a least-squares SVM model, but there exist many other SVM models that we have no good quantum algorithms for yet. The variety of classical SVM models and kernels makes solving a classification problem much easier in the classical case, so any further development in a different type of SVM quantum algorithm would be noteworthy.

Right now there is some development on a sparse quantum SVM that uses hinge loss and L_1 regularization instead of L_2 regularization sparse. The model is considered sparse because its classification will only depend on a few features. This model is very different from the least-squares SVM discussed here and can perform well on different types of problems than the least-squares SVM performs well on.

One other interesting open problem is dequantization. Dequantization is the process of trying to "de-quantize" a quantum algorithm by using its techniques to create a more efficient classical algorithm. The idea is that since the techniques used in quantum SVM are very different than those used in classical SVM, maybe there are some classical analogues to these techniques that could be synthesized to create a better classical algorithm. One paper proposes a quantum-inspired classical SVM algorithm that samples from the kernel matrix to make classifications based on estimations [19].

References

[1] C. Cortes and V. Vapnik. Support-vector network. Machine Learning, 20:273297, 1995.

[2] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors, Advances in Kernel Methods - Support Vector Learning, Cambridge, MA, 1998. MIT Press.

[3] Chang, Chih-Chung and Lin, Chih-Jen, LIBSVM: A Library for Support Vector Machines, ACM Transactions on Intelligent Systems and Technology, volume 2, issue 3, 2011.

[4] Philip E. Gill Elizabeth Wong, Methods for convex and general quadratic programming, Math. Prog. Comp. 7:71112, 2015.

[5] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research, 6:18891918, 2005.

[6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9, 1871-1874, 2008.

[7] T. Joachims. Text categorization with support vector machines: Learning with many relevant features, ECML, 1998.

[8] Patrick Rebentrost, Masoud Mohseni, Seth Lloyd, Quantum support vector machine for big data classification, Phys. Rev. Lett. 113, 130503 (2014). arXiv:1307.0471.

[9] S. Lloyd, M. Mohseni, and P. Rebentrost, Quantum algorithms for supervised and unsupervised machine learning, arXiv:1307.0411

[10] Davide Anguita, Sandro Ridella, Fabio Rivieccio and Rodolfo Zunino, Quantum optimization for training support vector machines, Neural Networks, Volume 16, Issues 56, JuneJuly 2003, Pages 763-770

[11] A. Harrow, A. Hassidim, and S. Lloyd, Quantum Algorithm for Linear Systems of Equations, Phys. Rev. Lett. 103, 150502 (2009)

[12] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du, Experimental Realization of a Quantum Support Vector Machine, Phys. Rev. Lett. 114, 140504

[13] V. Giovannetti, S. Lloyd, and L. Maccone, Quantum Random Access Memory, Phys. Rev. Lett. 100, 160501 (2008).

[14] J. Suykens and J. Vandewalle, Neural Process. Lett. 9, 293 (1999).

[15] S. Lloyd, M. Mohseni, and P. Rebentrost, Quantum principal component analysis, arXiv:1307.0401

[16] Berry, D. W.; Ahokas, G.; Cleve, R. and Sanders, B. C. Efficient quantum algorithms for simulating sparse Hamiltonians. Communications in Mathematical Physics, 2007, 270, 359-371.

[17] Lloyd, S.; Mohseni, M. and Rebentrost, P. Quantum self analysis. arXiv:1307.0401, 2013.

[18] Arodz, Tomasz and Saeedi, Seyran. Quantum Sparse Support Vector Machines. arXiv:1902.01879, 2019.

[19] Ding, Chen; Bao, Tian-Yi and Huang, He-Liang. Quantum-Inspired Support Vector Machine. arXiv:1906.08902, 2019

[20] Philip E. Gill and Elizabeth Wong. Methods for convex and general quadratic programming. Mathematical Programming Computation, 2015

[21] Quadratic Programming `https://www.math.uh.edu/~rohop/fall_06/Chapter3.pdf`

[22] Johan Suykens. Least Squares Support Vector Machines. NATO-ASI Learning Theory and Practice, 2002.