

Quantum SVM

Haley So and Huy Ha



The background is a solid orange color. In the top-left corner, there are three vertical bars of varying heights, each composed of several overlapping semi-transparent orange circles. In the bottom-right corner, there are four vertical bars of increasing height from left to right, each also composed of several overlapping semi-transparent orange circles.

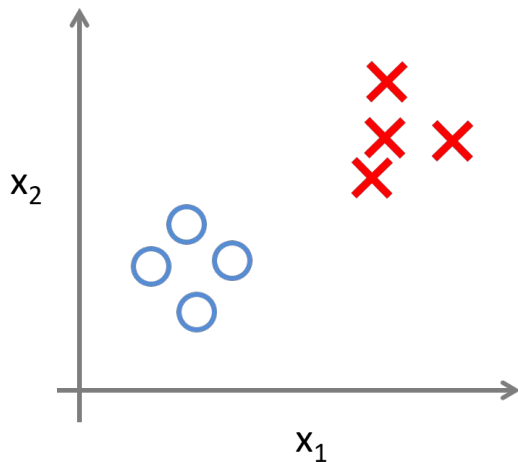
Classical SVM



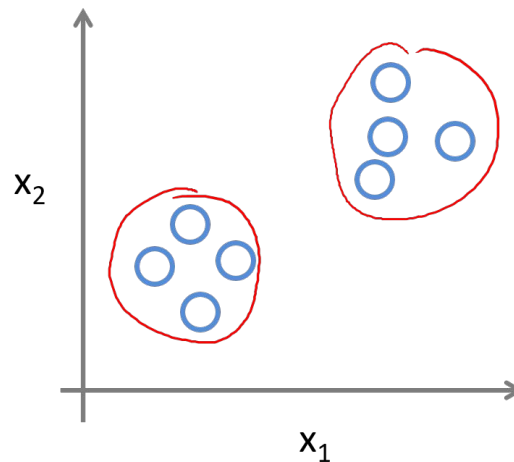
Quick intro to machine Learning

- Supervised vs unsupervised

Supervised Learning



Unsupervised Learning

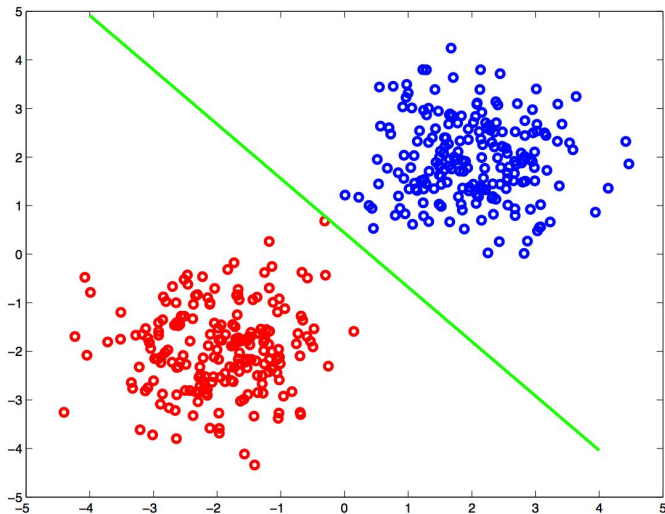


Classification into two groups is often beneficial



Classical SVM

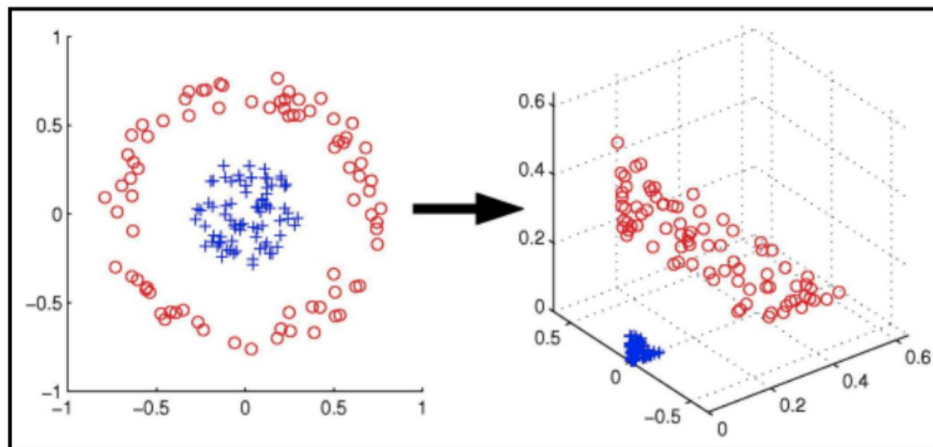
- A common supervised algorithm to classify is through **support vector machines (SVM)**
- If the data is **linearly separable**,
 - Find the “best” hyperplane/boundary





Classical SVM

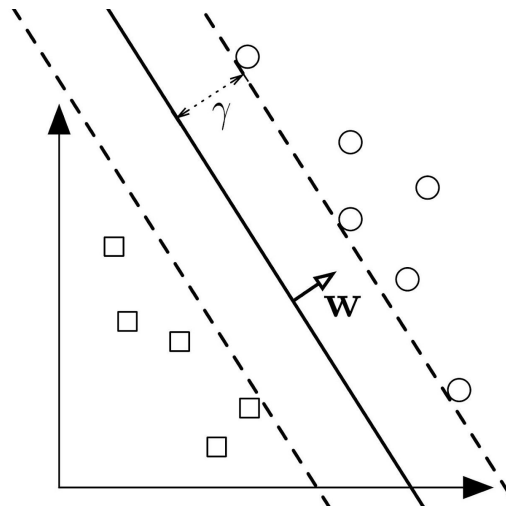
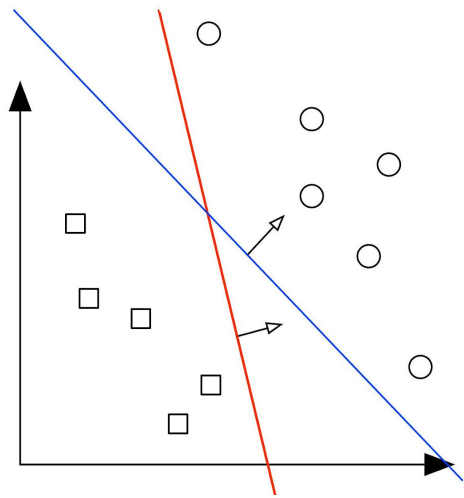
- A common supervised algorithm to classify is through **support vector machines (SVM)**
- Or, use the **kernel trick** to put our data in a **higher dimension** where it become linearly separable when you apply a function
 - Find the “best” hyperplane/boundary





Finding the boundary

- We can draw many boundaries, but one can argue that the best is one that maximizes the margin between the closest data point to the boundary



SVM finds the max-margin hyperplane that divides the two classes



Formulation of the problem

Given M training data points of the form

$$(\vec{x}_j, y_j) : \vec{x}_j \in \mathbb{R}^N, y_j = \pm 1$$

where $y_j = +1$ or -1

Let \vec{w} represent the norm to the **decision boundary**. The **margin** is given by two parallel hyperplanes, separated by distance $\frac{2}{\|\vec{w}\|}$ with no data points inside the margin

Formulation of the problem

The decision boundary of the two hyperplanes:

$$\vec{w} \cdot \vec{x} - b = +1$$

$$\vec{w} \cdot \vec{x} - b = -1$$

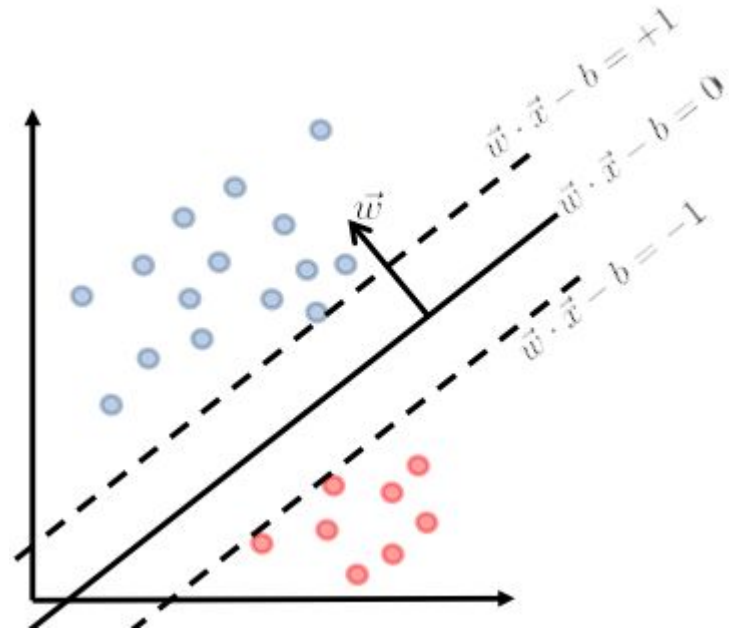
The training data is correctly classified if

$$\vec{w} \cdot \vec{x}_i - b \geq +1 \quad \text{if } y_i = +1$$

$$\vec{w} \cdot \vec{x}_i - b \leq -1 \quad \text{if } y_i = -1$$

So for all i ,

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq +1 \quad \text{for all } i$$





Formulation of the problem

Goal: maximize the distance between the two hyperplanes.

Maximize distance between the two hyperplanes: $\frac{2}{\|\vec{w}\|}$

Constraint: $y(\vec{w} \cdot \vec{x}_j - b) \geq 1$

SVM standard (primal) form (with slack):

$$\text{Minimize: } \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^n \xi_i$$

$$\text{Such that: } y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 - \xi_i$$

(for all i)

$$\xi_i \geq 0$$



Formulation of the problem

Since the problem is a quadratic program and the strong duality holds, we can take the dual.

The Lagrangian is: $L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2} \|\vec{w}\|^2 + \sum_{i=1}^n \alpha_i (1 - y_i(\vec{w} \cdot \vec{x}_i - b))$

Primal: $p^* = \min_{\vec{w}, b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha})$

$$\frac{\partial}{\partial \vec{w}} L(\vec{w}, b, \vec{\alpha}) = \vec{w} - \sum_{i=1}^n \alpha_i y_i \vec{x}_i \quad \Longrightarrow \quad \vec{w} = \sum_{i=1}^n \alpha_i y_i \vec{x}_i$$

$$\frac{\partial}{\partial b} L(\vec{w}, b, \vec{\alpha}) = \sum_{i=1}^n \alpha_i y_i \quad \Longrightarrow \quad \sum_{i=1}^n \alpha_i y_i = 0$$



Formulation of the problem

Since the problem is a quadratic program and the strong duality holds, we can take the dual.

The Lagrangian is:
$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2} \|\vec{w}\|^2 + \sum_{i=1}^n \alpha_i (1 - y_i(\vec{w} \cdot \vec{x}_i - b))$$

Primal:
$$p^* = \min_{\vec{w}, b} \max_{\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha})$$

Dual:
$$d^* = \max_{\alpha_i \geq 0} \min_{\vec{w}, b} L(\vec{w}, b, \vec{\alpha})$$

$$d^* = \max_{\alpha_i \geq 0} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j}^n \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

subject to
$$\sum_{i=1}^n \alpha_i y_i = 0$$



Formulation of the problem

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

Dual: maximize over the Karush-Kuhn-Tucker multipliers $\vec{\alpha} = (\alpha_1, \dots, \alpha_M)^T$

Subject to: $\sum_{j=1}^M \alpha_j = 0$ and $y_j \alpha_j \geq 0$.

The hyperplane parameters are recovered and only a few of the α_j 's are non-zero: these are the ones corresponding to the vectors that lie on the two hyperplanes.

$$\vec{w} = \sum_{j=1}^M \alpha_j \vec{x}_j \quad b = y_i - \vec{w} \cdot \vec{x}_j$$



Formulation of the problem

$$K_{jk} = k(\vec{x}_j, \vec{x}_k) = \vec{x}_j \cdot \vec{x}_k$$

is the Kernel matrix

The hyperplane parameters are recovered and only a few of the α_j 's are non-zero: these are the ones corresponding to the vectors that lie on the two hyperplanes.

$$y(\vec{x}) = \text{sgn} \left(\sum_{j=1}^M \alpha_j k(\vec{x}_j, \vec{x}) + b \right)$$

The result is a binary classifier for a new vector x . It will return -1 or +1.



Time

Solving the dual form involves evaluating the $M(M-1)/2$ dot products

$$\vec{x}_j \cdot \vec{x}_k$$

Finding the optimal α_j values takes $O(M^3)$.

Each dot product takes $O(N)$ to evaluate so the classical SVM algorithm takes

$$O(\log(\epsilon^{-1})poly(N, M))$$

With accuracy ϵ .

The background is a solid orange color. In the top-left corner, there are three vertical bars of varying heights, each composed of several overlapping semi-transparent orange circles. In the bottom-right corner, there are four vertical bars of increasing height from left to right, each also composed of several overlapping semi-transparent orange circles.

Quantum SVM



Why Quantum Machine Learning?

- Quantum computers are good at manipulating vectors and tensor products in high-dimensional spaces
- Classical data of N -dim complex vectors can be mapped onto quantum states of $\log_2 N$ qubits.
- Using qRAM, storing data takes $O(\log_2 N)$ steps
- Post-processing data in the quantum form takes $O(\text{poly}(\log N))$
- Evaluating distances between and inner products between large vectors takes less time in quantum than in the classical regime (exponentially hard)



Training





Assumptions

The paper assumes that **oracles** for the training data that return quantum vectors

$$|\vec{x}_j\rangle = \frac{1}{|\mathbf{x}_j|} \sum_{k=1}^N (\vec{x}_{jk}) |k\rangle$$

the norms $|\mathbf{x}_j|$ and labels y_j are given.

To efficiently construct these states, they use quantum RAM, which uses $O(MN)$ hardware resources but only $O(\log MN)$ operations to access them.



Kernel Matrix

- Use the inner product evaluation to prepare the kernel matrix.
- Played a crucial role in the dual formulation
- Least squares reformulation (to come!)

To prepare the normalized kernel matrix $\hat{K} = \frac{K}{\text{tr}(K)}$

1. Call the training data oracles with the state $\frac{1}{\sqrt{M}} \sum_{i=1}^M |i\rangle$

This prepares in quantum parallel the state $|\chi\rangle = \frac{1}{\sqrt{N_\chi}} \sum_{i=1}^M |\vec{x}_i||i\rangle|\vec{x}_i\rangle$

with $N_\chi = \sum_{i=1}^M |\vec{x}_i|^2$

Time: $O(\log NM)$



Kernel Matrix

- Use the inner product evaluation to prepare the kernel matrix.
- Played a crucial role in the dual formulation
- Least squares reformulation (to come!)

To prepare the normalized kernel matrix $\hat{K} = \frac{K}{tr(K)}$

If we discard the training set register, now we have the kernel matrix as a density matrix. They show it in the partial trace

$$tr_2\{|\chi\rangle\langle\chi|\} = \frac{1}{N_\chi} \sum_{i,j=1}^M \langle\vec{x}_j|\vec{x}_i\rangle |\vec{x}_i\rangle\langle\vec{x}_j| = \frac{K}{trK}$$

But, finding the inverse of K requires us to do exact exponentiation efficiently



Exponentiation

- Multiple copies of a quantum system with density matrix K can be used to construct the unitary transformation $e^{-i\hat{K}t}$
- As a result, one can perform quantum PCA, apply quantum phase algorithm to find the eigenvalues and eigenvectors of an unknown density matrix

<https://www.nature.com/articles/nphys3029>



Least-squares svm

The key idea is to employ the least-squares reformulation of the svm developed in “Least Squares Support Vector Machine Classifiers” by Suykens et al. So, instead of quadratic programming, the solution involves solving a set of **linear equations** to obtain the parameters.

Introduce slack variables e_j and replace the inequality constraints with equality constraints (using $y_j^2 = 1$):

$$y_j(\vec{w} \cdot \vec{x}_j + b) \geq 1 \longrightarrow (\vec{w} \cdot \vec{x}_j + b) = y_j - y_j e_j$$



Least-squares svm

The key idea is to employ the least-squares reformulation of the svm developed in “Least Squares Support Vector Machine Classifiers” by Suykens et al. So, instead of quadratic programming, the solution involves solving a set of **linear equations** to obtain the parameters.

The implied lagrange function contains a penalty term $\frac{\gamma}{2} \sum_{j=1}^M e_j^2$

where γ determines the relative weight of training error

Taking partial derivatives of the lagrange function and eliminating the variables \vec{u} and e_j leads to a least-squares approximation of the problem



Least-squares approximation

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

where $K_{ij} = \vec{x}_i^T \cdot \vec{x}_j$ is the symmetric kernel matrix, $\vec{y} = (y_1, \dots, y_M)^T$ and

$$\vec{1} = (1, \dots, 1)^T$$

F is $(M+1) \times (M+1)$ dimensional. The additional row and column arise because of the non-zero offset b .

The α_j take on the role as distance from the optimal margin.



The SVM Parameters

The SVM parameters are determined by: $(b, \vec{\alpha}^T)^T = F^{-1}(0, \vec{y}^T)^T$

We generate the state $|b, \vec{\alpha}\rangle$ that describes the hyperplane with the matrix inversion algorithm to classify a state $|\vec{x}\rangle$

The classifier will be determined by the success probability of a swap test between $|b, \vec{\alpha}\rangle$ and $|\vec{x}\rangle$

For the quantum matrix inversion algorithm, \hat{F} needs to be exponentiated efficiently. (Lie product formula)

Phase estimation generates a state which is close to the ideal state storing the respective eigenvalues.

Expansion coefficients of the new state are the desired svm parameters $C = b^2 + \sum_{k=1}^M \alpha_k^2$

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



Classification





Result of Training and Goal of Classification

- The result of training is a quantum state that encodes the offset and weight contributions of each support vector (in this case, it's usually all training data)

$$|b, \vec{\alpha}\rangle = \frac{1}{\sqrt{C}} \left(b|0\rangle + \sum_{k=1}^M \alpha_k |k\rangle \right). \quad (C = b^2 + \sum_{k=1}^M \alpha_k^2)$$

- Now, given a new data instance, encoded as a quantum state $|\vec{x}\rangle$, we want to classify it as either -1 or +1.

Swap Test

between two quantum states constructed from the hyperplane quantum state and the new query state



Algorithm for Classification

Preparing to measure:

1. Construct $|\tilde{u}\rangle$ and $|\tilde{x}\rangle$
2. Using an ancilla, construct $|\psi\rangle$ and $|\phi\rangle$
3. Measure ancilla

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

Measurement to answer:

1. To obtain P with accuracy ϵ , repeat measurement $O(P(1-P)/\epsilon^2)$ times.
2. If $P < 0.5$, then $|\vec{x}\rangle$ belong to +1 class. Else, -1.

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

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\tilde{u}\rangle + |1\rangle|\tilde{x}\rangle)$$

$$|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$P = |\langle\psi|\phi\rangle|^2 = \frac{1}{2}(1 - \langle\tilde{u}|\tilde{x}\rangle)$$



Low Rank Approximation of Kernel Matrix

- In order to calculate the inverse of F for solving least squares, we need to find eigenvalues and eigenvectors of F .

$$F \equiv \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & K + \gamma^{-1} \mathbb{1} \end{pmatrix}$$

- Eigenvalue of F at most 1 (F is normalized), and minimum less than or equal to $O(1/M)$
 - (Training examples with no overlap with other training examples)
- Therefore, condition number is $O(M)$
 - Exponential runtime [Quantum Algorithm for solving linear systems of equations]
- Define a ϵ_K , such that only eigenvalues larger than ϵ_K is considered
 - Filtering process in [Quantum Algorithm for solving linear systems of equations] takes into account the effective condition number $1/\epsilon_K$.



Low Rank Approximation of Kernel Matrix

- From data matrix $X = (\vec{x}_1, \dots, \vec{x}_M)$, construct the kernel matrix and covariance matrix

$$K = X^T X \quad \Sigma = X X^T$$

- These matrices have the same non zero eigenvalues
- PCA admits low rank approximation of covariance matrix by considering eigenvectors corresponding to largest eigenvalues



Performance Analysis





Classical SVM

- $O(\log(1/\epsilon) M^2(N+M))$:
 - Need to evaluate the $M(M-1)/2$ dot products, each taking $O(N)$
 - Find optimal alpha values by quadratic programming ($O(M^3)$ in non sparse case)



Quantum SVM

- $O(\kappa_{\text{eff}}^3 \epsilon^{-3} \log MN)$
 - κ_{eff} is the effective condition number of value $1/\epsilon_K$ with the smallest eigenvalue considered is ϵ_K
 - ϵ is error from matrix inversion and phase estimation
 - $\log MN$ comes from kernel matrix preparation
- In summary, QSVM scales as $O(\log MN)$.
 - Therefore, quantum advantage in $O(\text{poly } M)$ training examples and $O(N)$ samples for inner product is required.

Nonlinear QSVM



Feature mapping to higher dimensional feature space

- Quantum Computers can efficiently manipulate high dimensional vectors
 - Good candidates for polynomial kernel machines
- Consider simple feature map of d-times tensor product $|\phi(\vec{x}_j)\rangle \equiv |\vec{x}_j\rangle \otimes \dots \otimes |\vec{x}_j\rangle$, so the kernel function between x_j and x_k becomes $\langle\phi(\vec{x}_j)|\phi(\vec{x}_k)\rangle = \langle\vec{x}_j|\vec{x}_k\rangle^d$
- A linear hyperplane optimization in d-times tensor product space becomes a nonlinear surfaces in the original space.




Demo!



Extra Slides





$$\hat{F}|b, \vec{\alpha}\rangle = |\vec{y}\rangle$$

The right-hand side $|\vec{y}\rangle$ can be formally expanded into eigenstates $|u_j\rangle$ of \hat{F} with corresponding eigenvalues λ_j , $|\vec{y}\rangle = \sum_{j=1}^{M+1} \langle u_j | \vec{y} \rangle |u_j\rangle$. With a register for storing an approximation of the eigenvalues (initialized to $|0\rangle$), phase estimation generates a state which is close to the ideal state storing the respective eigenvalue:

$$|\vec{y}\rangle|0\rangle \rightarrow \sum_{j=1}^{M+1} \langle u_j | \vec{y} \rangle |u_j\rangle |\lambda_j\rangle \rightarrow \sum_{j=1}^{M+1} \frac{\langle u_j | \vec{y} \rangle}{\lambda_j} |u_j\rangle.$$

The second step inverts the eigenvalue and is obtained as in [16] by performing a controlled rotation and uncomputing the eigenvalue register. In the basis of training set labels, the expansion coefficients of the new state are the desired support vector machine parameters: ($C = b^2 + \sum_{k=1}^M \alpha_k^2$),

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



Haley:

- Classical SVM and problem formulation
- Classical SVM Demo
- Quantum Training

Huy:

- Quantum Classification
- Results and Speed ups
- QSVM implementation Demo