Des-q: a quantum algorithm to provably speedup retraining of decision trees

Niraj Kumar[†*], Romina Yalovetzky[*], Changhao Li, Pierre Minssen, and Marco Pistoia

Global Technology Applied Research, JPMorgan Chase, New York, NY 10017 USA

Decision trees are widely adopted machine learning models due to their simplicity and explainability. However, as training data size grows, standard methods become increasingly slow, scaling polynomially with the number of training examples. In this work, we introduce Des-q, a novel quantum algorithm to construct and retrain decision trees for regression and binary classification tasks. Assuming the data stream produces small, periodic increments of new training examples, Des-q significantly reduces the tree retraining time. Des-q achieves a logarithmic complexity in the combined total number of old and new examples, even accounting for the time needed to load the new samples into quantum-accessible memory. Our approach to grow the tree from any given node involves performing piecewise linear splits to generate multiple hyperplanes, thus partitioning the input feature space into distinct regions. To determine the suitable anchor points for these splits, we develop an efficient quantum-supervised clustering method, building upon the q-means algorithm introduced by Kerenidis *et al.* We benchmark the simulated version of Des-q against the state-of-the-art classical methods on multiple data sets and observe that our algorithm exhibits similar performance to the state-of-the-art decision trees while significantly speeding up the periodic tree retraining.

1 Introduction

We are currently in the era of *big-data* where organizations collect vast amounts of daily user data [54], be it with retail chains logging transactions, telecommunication companies connecting millions of calls and messages, or large banks processing millions of ATM and credit card transactions daily [67, 57]. To tackle the data surge, entities aim to build efficient, accurate machine learning models and swiftly retrain them to avoid performance degradation. As the Internet expands, data volume grows, challenging current methods.

Despite the success of deep neural network models [55] for these tasks, decision tree-based models for regression and classification remain competitive and widely adopted [65, 15, 63]. A decision tree is a logical model comprising a hierarchical set of rules for predicting target labels based on training examples. They are favored for their model explainability, low parameter requirements, noise robustness, and efficiency with large-scale datasets, offering a cost-effective alternative to other supervised learning models [56]. Despite its practical success, the construction of an optimal decision tree from a given training dataset is considered to be a hard problem [23, **?**]. Consequently, heuristics such as greedy methods have long been used to construct trees. These top-down or bottom-up recursive inducers seek to find the best local feature and threshold value to partition the data. Popular methods following this approach include CART [10, 36, 13], ID3 [50], and C4.5 [51]. Building a decision tree model using these methods for a training dataset with N examples and d features incurs a time complexity that scales polynomially in both N and d.

In practical applications, decision trees require periodic updates with batches of new labeled data to prevent model deterioration over time. These updates fall into two categories: *incremental decision tree* methods, which update the tree using only new data [61, 16, 43], and *tree retraining* methods, which rebuild the tree using both old and new data. Incremental methods offer quick updates but struggle with concept drift, where

Niraj Kumar[†*]: †niraj.x7.kumar@jpmchase.com., ∗: NK and RY contributed equally in this work.

model performance degrades as data distributions change [60]. Therefore, retraining the entire tree is often more effective, though existing methods make it time-consuming and resource-intensive, posing challenges in big-data environments.

Parallel to classical development, efforts have aimed to build quantum decision trees using techniques like Grover's search algorithm for potential speedups [42, 34, 5, 26]. Despite these attempts, efficient tree construction and retraining remain challenging, achieving at best a quadratic speedup in the number of features d over classical methods. However, these methods do not offer speedups in the number of training examples N. Since in most practical cases the number of training examples greatly exceeds the number of features, a significant speedup is only meaningful if achieved over N. This limitation becomes more pronounced with periodic data accumulation, where timely model retraining is essential to prevent performance degradation. Therefore, a critical question arises: *once the decision tree model is built and put online, is it possible to achieve a poly-logarithmic in N run-time dependence in the decision-tree retraining?*[1] Achieving this goal would ensure the model remains consistently online, with rapid retraining capabilities and sustained performance quality.

In this work, we answer this question affirmatively by proposing a novel quantum algorithm called Des-q for decision tree construction and retraining for regression and binary classification. The initial tree construction takes time polynomial in N and d solely due to the time to load the initial data into quantum-accessible-data structure[2]. In this work, we utilize the KP-tree data structure [33], which, once constructed, allows for querying data as quantum superposition states in polylogarithmic time relative to N and d. Moreover, this permits the rest of the algorithmic steps of Des-q to also run in polylogarithmic time. Although tree construction with Des-q is initially slow due to KP-tree creation for data with N examples, subsequent retrains with small batches of new data become extremely fast. This process simply involves updating the KP-tree with the new small batch of data while the remaining algorithmic steps are executed in poly-logarithmic time relative to N.

The rest of the paper is structured as follows. We review the related works in Sec 2, while Sec 3 provides a key motivation for Des-q including a high-level sketch of its methodology. Our key results are summarized in Sec 4. Sec 5 provides the detailed algorithmic steps along with key theorems for tree construction with Des-q, while Sec 6 provides the steps for tree retraining. We provide the benchmark for simulated Des-q in Sec 7 and finally conclude in Sec 8. We also highlight some ingredients we utilize from existing quantum algorithms in Appendix A.

1.1 Terminology used in the paper

1. *Root node*: no incoming edge, zero or more outgoing edges; *internal node*: one incoming edge, two (or more) outgoing edges; *leaf node*: each leaf node is assigned a class label; *parent and child node*: if a node is split, we refer to that given node as the parent node, and the resulting nodes are called child nodes, respectively

2. Throughout this work, we refer to

   - N: The number of training examples in the dataset
   - d: The number of attributes/features in the dataset
   - D: Maximum Tree height/depth
   - k: Number of clusters generated at each clustering step
   - K: Maximum allowed number of iterations in each clustering step
   - N_{new}: Number of new training examples for periodic model retrain

3. The training dataset is referred as $\mathsf{Data} = \{X, Y\}$, where $X = [x_1, \cdots, x_N]^T \in \mathbb{R}^{N\times d}$ and $Y = [y_1, \cdots, y_N]^T$. X contains the N training examples $x_i \in \mathbb{R}^d$, i.e., consisting of d features with only numerical values, and Y is the target label with each $y_i \in \mathbb{R}$ for the task of regression, while $y_i \in \mathcal{M} = \{0, 1\}$ for the task of binary classification.

[1] This implies an exponential speedup in tree retraining over classical greedy methods, although further improvements could occur in classical methods from quantum-inspired classical techniques [58, 17].

[2] By quantum-accessible data structure, we mean a classical data structure that allows accessing quantum states of the input data in poly-logarithmic time relative to the input size. Constructing this data structure requires polynomial time relative to the input size.

2 Related work

Beginning with classical decision trees, the state-of-the-art decision tree methods to handle both classification and regression tasks include CART [10, 36], ID3 [50] and C4.5 [51]. It is worth noting that, decision trees built using these methods for a dataset with N instances and d features typically involve a computational complexity of $O(poly(Nd))$. Beyond heuristic methods, approaches to train optimal decision trees using mixed-integer programs (MIP) have been proposed [66, 18]. Alternative techniques for constructing decision trees involve utilizing clustering mechanisms have also been proposed [38, 6].

On the quantum approaches to construct tree, the work by Lu *et al.*[41] encodes the training data into quantum states and leverages Grover's search algorithm to build the tree. In the work by Khadiev *et al.* [34], the classical decision tree induction algorithm $C5.0$ is extended to a quantum version using Grover search-based algorithm. The quantum algorithm has a run-time complexity that is nearly quadratic better than the classical algorithm in terms of feature number d. Heese *et al.* [26] introduce a quantum representation of binary classification trees called *Q-tree*. The tree is constructed through probabilistic traversal via quantum circuit measurements.

We also remark on the two recent papers on improving the existing q-means clustering algorithm [31] whose modified version is a component of our Des-q algorithm. The first paper by Jaiswal [29] where the author provides a quantum algorithm for k-means problem with provable convergence guarantees, and Doriguello *et al.* [17] who improve upon the existing q-means algorithm while still being heuristic.

3 Motivation of our method

As highlighted by Ho *et al.* [27], numerous approaches exist for building decision trees from training examples. Most methods use linear splits at each internal node to create branches, partitioning examples with one or more hyperplanes into distinct regions of the feature space. These linear splits fall into three categories: *axis-parallel linear split, oblique linear split, and piecewise linear split.*

Axis-parallel linear splitting divides data based on a specific feature l and a threshold θ_l, determined by minimizing an impurity metric like Gini impurity, entropy (for classification), or label variance (for regression). Subsequently, each example $x_i \in \mathbb{R}^d$, $i \in [N]$ is assigned to a branch based on whether its feature value x_{il} exceeds θ_l. If $x_{il} \leq \theta_l$, the example goes to one branch; otherwise, it goes to the other. Popular decision tree methods like ID3, C4.5, and CART use this splitting mechanism.

The oblique linear split method divides data by finding a hyperplane in the feature space, not necessarily parallel to any axis. Each example is assigned based on which side of the hyperplane it lies on, determined by the condition

$$\sum_{l=1}^{d} a_l x_{il} \leq \theta,$$

where a_l is the coefficient of the l-th feature and θ is the threshold. This method often produces lower-depth trees with better generalization but finding the optimal hyperplane is challenging. Proposed solutions include using simulated annealing or hill-climbing to approximate near-optimal hyperplanes [13, 11].

The piecewise linear split method extends the oblique linear split by using multiple hyperplanes to create $k(\geq 2)$ branches or regions. Each hyperplane is defined by anchor points, and examples are assigned based on their proximity to these points. Although selecting these hyperplanes is more complex than the oblique method, it can significantly reduce tree depth and improve generalization, especially when multiple features are strongly correlated.

In particular, Liu *et al.* [38] proposed using the piecewise linear split technique to create a Multi-Split Decision Tree (MSDT) for multi-class classification. In MSDT, each branch acts as a cluster with an anchor point as its centroid. They first calculate feature weights using the RELIEF-F algorithm [35] to determine feature relevance. Then, they use weighted k-means [40] to cluster examples, forming k clusters. This weighted approach ensures nodes generated by the supervised k-means minimize node class impurity by giving more weight to features strongly correlated with the target label, aligning with the decision tree's objective to minimize class impurity or variance.

Our approach to constructing decision trees builds on a similar concept, involving piecewise

linear splits using a supervised clustering quantum algorithm.

4 Summary of results

We introduce the Des-q algorithm for tree construction in Algorithm 1 along with its illustration in Figure 1. Given the dataset Data $= \{X, Y\}$ introduced in Sec 1.1, we use a novel supervised quantum clustering algorithm that leverages the unsupervised quantum clustering algorithn named q-means introduced by Kerenidis *et al.* [31]. With this supervised quantum clustering algorithm, we generate k nodes, or clusters, from the root node. For this, we pick the k initial centroids $\{c_1^0, \cdots c_k^0\} \in \mathbb{R}^d$. Here the superscript denotes the centroids at the 0-th iteration. At each t-th iteration, for each training sample $\{x_i\}_{i\in[N]}$, its cluster label is assigned. The criterion utilized leverages the idea proposed by Liu *et al.* [38] mentioned before where the cluster assignment is based on the weighted Euclidean distance that incorporates the feature weights $w_j \in [0, 1]$ of each feature vector $j \in [d]$ that measure the statistical significance of each feature vector of X in predicting the target label vector Y. In Des-q we utilize the normalized absolute value of Pearson correlation coefficient [48] for regression and the point-biserial correlation [21], which is a special case of Pearson correlation for binary label datasets, for classification. These feature weights are estimated with the proposed quantum methods. By doing this, the cluster labels are assigned as

$$\textit{cluster-label}_i^t = \arg\min_{1\le l\le k} \|x_i - c_l^t\|_w, \qquad (1)$$

where $\|x_i - c_l^t\|_w = \sqrt{\sum_{j=1}^d w_j \cdot (x_{ij} - c_{lj}^t)^2}$ is the weighted distance[3].

By utilizing this weighted distance, we are making the clustering algorithm supervised as it incorporates some information from the labels. We will show in Section 7 how incorporating these weights improved the accuracy performance. After all the training examples are partitioned in the k disjoint clusters $\mathcal{C}_l^t$, $l \in [k]$, each cluster centroid c_l^t is updated as the mean of the examples in the cluster. The above two steps are repeated until some stopping criteria. At this point, the first depth of the decision tree has been constructed, where the k nodes correspond to k clusters and these nodes are represented by their centroids. The tree keeps growing using the mentioned supervised quantum clustering technique until it reaches a maximum depth D. Next, we need to extract the label information from each of the leaf nodes $l \in [k^D]$, which are characterized by the clusters $\{\mathcal{C}_l\}$. Our proposal is to use Des-q to train (and retrain) the decision tree and then perform inference on the classical computer. To perform classical inference on an unseen sample, the algorithm traverses the tree by selecting nodes with the smallest weighted distance. Once it reaches a leaf node, it uses the label information extracted from that leaf node to assign a label.

We provide a broad overview of our Des-q algorithm for tree construction and retraining and its steps in the subsections below. Subsequently, in Sec 5, we delve into each of these components, offering detailed insights into their algorithmic steps and discussion on their algorithmic complexities.

4.1 Des-q for tree construction

We now explain each step of Des-q and their time complexities.

Step 1 [*Loading classical data*]: We consider the dataset Data as introduced in Section 1.1. This dataset is stored within a specialized tree-like data structure known as the *KP-tree* [33]. Leveraging quantum superposition access to this data structure enables us to prepare *amplitude-encoded states* for the training examples, feature vectors, and label vector[4]. It is worth noting that constructing the KP-tree initially requires a time of $T_{kp} = \tilde{\mathcal{O}}(Nd)$. Once constructed, querying the quantum states associated with the example vectors (in superposition) can be done in $\mathcal{O}(poly\log(Nd))$ time, while the query time for the superposition state of target labels is $\mathcal{O}(poly\log(N))$.

[3]Note that this is different to unsupervised k-means where $\|x_i - c_l\| = \sqrt{\sum_{j=1}^d (x_{ij} - c_{lj})^2}$ is used for cluster label assignment.

[4]We query the KP-trees via quantum superposition of the indices and get the quantum superposition state corresponding to the index and the amplitude encoded state corresponding to the example stored in that indexed KP-tree.

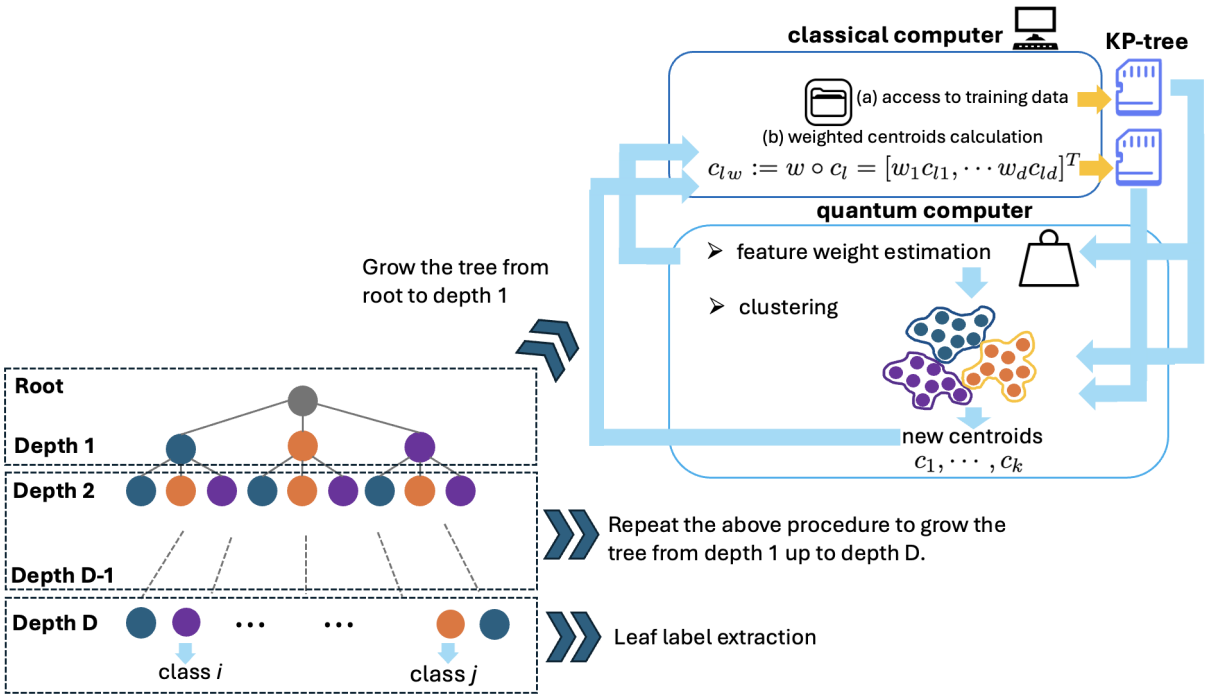


Figure 1: Diagram of Des-Q. We highlight the procedure to construct a decision tree from root node to depth D. The yellow arrows indicate communication between classical components whereas the light blue arrows indicate between classical and quantum components. To grow the tree from root to depth 1, we load the data into the KP-tree data structure from which the samples are queried in superposition to the quantum computer, and the feature weights are estimated. Subsequently, one performs weighted (supervised) clustering to generate k clusters corresponding to depth 1. The above procedure is repeated to grow the tree up to depth D. Finally, we perform the leaf label extraction where we assign classes to the leaf nodes.

Step 2 [*Feature weight estimation*]: We quantumly estimate the strength of each feature vector in predicting the label value using the Pearson correlation coefficient and refer to this as feature weight. The Pearson correlation can be applied to assess the strength of features not only when the labels are numerical, as in regression, but also in the context of binary classification by applying the special case of Pearson correlation known as point-biseral. For each feature, we perform this estimation efficiently with amplitude-encoded quantum states using the Frobenius Test circuit [32] in combination with the amplitude amplification step [8], where the circuit runtime scales as $\mathcal{O}(\log N/\epsilon)$, where ϵ is the estimation error.

Step 3 [*Supervised Clustering*]: We adapt the efficient (in N) unsupervised quantum clustering technique of Kerenidis *et al.* [31] called *q-means* for the purpose of constructing decision trees. This adaptation involves performing clustering on the input dataset using weighted distance estimation. The central idea is to pick k initial centroids, perform the Hadamard multiplication between the centroids and the weight vector to create weighted centroids, and finally store them in the KP-tree to access them as amplitude-encoded states[5].

Subsequently, we perform coherent quantum operations to estimate the inner product of all the data points with each of the k weighted centroids. We call this procedure weighted inner product estimation. This is followed by quantumly finding the centroid having maximum inner product value with the example vector in order to assign the *centroid-label* to that example. This corresponds to the closest centroid for that example. This creates a superposition state of the index of the example and their associated centroid-label $\in \{1, \cdots, k\}$ which indicates which cluster have they been assigned to. Next, we proceed to update the k centroid states.

[5]Once the data is loaded in KP-tree, the clustering process itself is efficient in N, meaning its runtime depends poly$-$logarithmically in the number of examples (N).

Algorithm 1 Tree construction with Des-q

Require: Data $= \{X, Y\}$, where $X = [x_1, \cdots, x_N]^T \in \mathbb{R}^{N\times d}$ and $Y = [y_1, \cdots, y_N]^T$. Each $y_i \in \mathbb{R}$ for regression task, and $y_i \in \mathcal{M} = \{0, 1\}$ for binary classification. X can alternatively be written as $X = [x^{(1)}, \cdots, x^{(d)}]$ where $x^{(j)}$ are feature vectors of length N; D: maximum tree depth; Error parameters ϵ_1 for feature weight estimation, and ϵ_2 for errors in clustering; Precision parameter δ for quantum clustering; k: number of clusters; K: maximum number of clustering iterations.

Ensure: The structure of the decision tree, the centroids $\{c_{\text{node}} \in \mathbb{R}^d\}$ corresponding to each internal node in the tree, and the leaf label prediction.

Step 1: Load X and Y using quantum-accessible-data structure [33].

X row access: $\tilde{U}_x : |i\rangle|0\rangle \to |i\rangle \otimes \frac{1}{\|x_i\|}\sum_{j=1}^{d} x_{ij}|j\rangle = |i\rangle|x_i\rangle$

Y column access: $\tilde{U}_y : |i\rangle|0\rangle \to |i\rangle \otimes \frac{1}{\|y\|}\sum_{i=1}^{N} y_i|i\rangle = |i\rangle \otimes |Y\rangle$

X column access: $\tilde{U}_{x^{(j)}} : |0\rangle|j\rangle \to \frac{1}{\|x^{(j)}\|}\sum_{i=1}^{N} x_{ij}|i\rangle \otimes |j\rangle = |x^{(j)}\rangle \otimes |j\rangle$

Step 2: SWAP test for feature weight estimation. Use SWAP test followed by amplitude amplification between $|x^{(j)}\rangle$ and $|Y\rangle$ states to estimate the Pearson correlation coefficient between each feature $x^{(j)}$ and label vector Y

$$w_j = \frac{\sum_{i=1}^{N}(x_{ij}-\mu_j)(y_i-\mu_y)}{\sqrt{\sum_{i=1}^{N}(x_{ij}-\mu_j)^2}\sqrt{\sum_{i=1}^{N}(y_i-\mu_y)^2}},$$

where $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$, $\mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i$ and $w := \{w_1, \cdots, w_d\}$ is referred as feature weight vector.

Step 3: Quantum supervised clustering to generate depth-1 tree.

while $t \leq K$**:**

3.1: Given k centroids $c_1, \cdots, c_k$ at iteration t. Perform Hadamard multiplication between w and each c_j and store the resulting weighted centroid in quantum-accessible-data structure.

3.2: **Perform weighted centroid distance estimation.** Query X and weighted centroid as quantum states to perform the mapping

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]} |j\rangle|0\rangle \to \frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]} |j\rangle|I_w(x_i, c_j)\rangle,$$

where $I_w(x_i, c_j)$ is the estimated weighted inner product between x_i and the centroid c_j.

3.3: **Assign the X examples to clusters**. Find the maximum inner product value $I_w(x_i, c_j)_{j\in[k]}$ for each example x_i and create the superposition of all points and their new assigned *centroid labels* (based on maximum inner product) as

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle \otimes_{j\in[k]} |j\rangle|I_w(x_i, c_j)\rangle \to \frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle|\textit{centroid-label}_i\rangle,$$

where $\textit{centroid-label}_i \in [k]$ is the centroid label assigned to $|x_i\rangle$.

3.4: **Update centroids**. Measure the second register of the previous state to obtain the characteristic vector state as

$$|\xi_j\rangle = \frac{1}{\sqrt{|\mathcal{C}_j|}}\sum_{i\in\mathcal{C}_j}|i\rangle \quad \forall j \in [k].$$

The updated centroid vector c_j is simply the average of the values assigned in the cluster j. In order to do this, estimate the inner product of the feature vector $|x^{(l)}\rangle$ and the characteristic vector $|\xi_j\rangle$ to estimate the l^{th} component of the j^{th} updated centroid vector. This is repeated for all $j \in [k], l \in [d]$.

end while

Step 4: Grow the tree to depth D. The clusters generated in the previous step correspond to the depth 1 of the tree. To further grow the tree, for each current node, generate a superposition of examples in the node cluster. Perform supervised clustering on the cluster examples to grow the tree until it reaches a maximum set depth D.

Step 5: Leaf label assignment. For each leaf node/cluster $j \in [k^D]$, perform the fidelity estimation between the state $|Y\rangle$ and $|\xi_j\rangle$ to obtain the mean label value $\textit{label}_j = \frac{1}{\sqrt{|\mathcal{C}_j|}}\sum_{i\in\mathcal{C}_j} y_i$. For regression, the mean label value is the leaf label. For the binary classification, if the mean is less than 0.5, the leaf is assigned a label 0, and 1 otherwise.

Measurement operations on the second register of the superposition state result in obtaining the k *characteristic states*, where the j^{th} characteristic state is a superposition of the indices corresponding to the examples assigned to the j^{th} cluster. The updated centroid is the average of the values of the elements in the cluster. This averaging is done by performing quantum multiplication of the transposed data matrix with the characteristic vectors. In contrast to the approach in [31], where direct matrix multiplication is followed by quantum state tomography to obtain the updated centroid quantum state, our method involves estimating centroid vectors directly through inner product estimation between the weighted data column quantum states and the characteristic vector states. The clustering procedure is then repeated either a maximum of K times or until convergence is achieved. This process results in the creation of the depth 1 of the decision tree with k child nodes. The computational time for this operation is $\mathcal{O}(poly\log(Nd))$, with other terms omitted for simplicity.

Step 4 and 5 [*Tree Growth and Majority Label*]: The growth of the tree continues through iterative clustering of the elements within the nodes at the previous depth, persisting until the desired tree depth D is achieved. The overall runtime for constructing the decision tree up to this stage is $\mathcal{O}(poly\log(Nd))$.

Subsequently, labels are assigned to each leaf node. For regression tasks, the label corresponds to the mean of the label values of the examples within the cluster. In binary classification, the label is set to 0 if the mean value is less than 0.5 and 1 otherwise. This is performed by quantumly estimating the mean value where the mean value can be obtained from the inner product of the characteristic vector state corresponding to the leaf node and the label vector state $|Y\rangle$. We show that the estimation of leaf label can be done efficiently with complexity $\mathcal{O}(poly\log(Nd))$.

The overall time complexity for constructing the initial decision tree is the sum of two components: the initial time required for loading the examples into the KP-tree data structure and the time spent on the actual tree construction. It becomes evident that while the tree-building process itself takes $\mathcal{O}(poly\log(Nd))$ time, it is the initial data loading into the KP-tree structure that causes the initial tree construction to scale $\tilde{\mathcal{O}}(Nd)$. However, as elaborated in the subsequent section, tree retraining, a central task in supervised learning models, proves to be efficient in terms of N since we retrain with new batches containing significantly fewer training samples.

4.2 Des-q for tree retraining

Once the tree has been initially constructed and deployed, it needs to be periodically retrained to account for new batch of training data of size N_{new}. Frequently, the quantity of new training examples is substantially smaller than the initial training set size, i.e., $N_{\text{new}} \ll N$. This periodic step is of crucial importance for the model, serving to prevent performance deterioration and maintain consistent model performance.

The retraining process begins by querying quantum superposition states from the KP-tree, encompassing the entire set of $N_{\text{tot}} = N + N_{\text{new}}$ training examples. Since the original N examples were already preloaded into the KP-tree during the tree construction part, one is only required to load the new examples into the KP-tree, a step that consumes time $\tilde{\mathcal{O}}(N_{\text{new}}d)$. Subsequently, as highlighted in Algorithm 2, we replicate all the stages of the Algorithm 1 designed for constructing the tree, but where we instead query superposition states over the set of N_{tot} examples. Upon loading the new training data into the data structure, the Des-q algorithm for retraining has a runtime of $\mathcal{O}(poly\log(N_{\text{tot}}d)) \approx \mathcal{O}(poly\log(Nd))$. Thus we see that the total time to build the tree is the time to load the new examples into the KP-tree followed by the time to run the algorithm

$$\tilde{\mathcal{O}}(N_{\text{new}}d) + \mathcal{O}(poly\log(Nd)).$$

Under the small batch updates criterion, we see that our algorithm for tree retraining scales polylogarithmically in N, thus ensuring that the tree retraining process is swift and efficient.

Algorithm 2 Tree retraining with Des-q

Require: Data $= \{X_{\text{tot}}, Y_{\text{tot}}\}$, where $X_{\text{tot}} = X : X_{\text{new}} = [x_1, \cdots, x_N, \cdots, x_{N_{\text{tot}}}]^T \in \mathbb{R}^{N_{\text{tot}} \times d}$ and $Y_{\text{tot}} = Y : Y_{\text{new}} = [y_1, \cdots, y_N, \cdots y_{N_{\text{tot}}}]^T$. The rest of the requirements are the same as Algorithm 1.

Ensure: the structure of the decision tree, the centroids $\{c_{\text{node}} \in \mathbb{R}^d\}$ corresponding to each internal node in the tree, and the leaf label prediction.

Step 1: load X_{new} and Y_{new} using quantum-accessible-data structure [33].

Step 2: Repeat steps 2:5 of Algorithm 1 but with querying superposition states corresponding to X_{tot} and Y_{tot}.

5 Tree construction with Des-q

Now that we have presented the high-level description of Des-q, we present the technical details of each component of the decision tree construction for the tasks of regression and binary classification. We also utilize other quantum algorithmic ingredients in Des-q, for which we refer the reader to Appendix A. The five key components of Des-q for tree construction are

1. *Data loading:* Load the classical data into quantum amplitude states.
2. *Feature weight estimation:* Estimate the Pearson correlation between each feature vector and the target label.
3. *Supervised clustering:* Perform supervised clustering at root node to generate nodes at first depth.
4. *Tree growth:* For each node, generate the superposition of the data examples and perform supervised clustering on the superposition state to further grow the tree until reaching maximum set depth D.
5. *Leaf label assignment:* When the leaf node is reached, compute the mean of the label values of the examples in the cluster. For regression, the leaf label is the mean, while for binary classification, the leaf label is 0 if the mean is less than 0.5, and 1 otherwise.

The subsections below correspond to each of the above-mentioned components.

5.1 Data loading

The first step to build a quantum algorithm on classical dataset Data $= \{X, Y\}$ (Sec 1.1) is to be able to load it into a quantum accessible memory such that one can efficiently query the data as quantum encoded states. Among the most space-efficient quantum encoding proposals is the amplitude encoding scheme which provides a natural link between quantum computing and linear algebra to be able to construct useful quantum algorithms [45].

5.1.1 Quantum amplitude encoding

The quantum amplitude encoding requires $\lceil \log(d) \rceil$ qubits to encode a vector $x = (x_1, \cdots, x_d) \in \mathbb{R}^d$ using the following form

$$|x\rangle = \frac{1}{\|x\|} \sum_{j=1}^{d} x_j |j\rangle . \tag{2}$$

Further one can also encode N vectors simultaneously, which is equivalent to encoding a matrix $X \in \mathbb{R}^{N \times d}$ as

$$|X\rangle = \frac{1}{\|X\|_F} \sum_{i=1}^{N} \|x_i\| |x_i\rangle |i\rangle , \tag{3}$$

where $|x_i\rangle = \frac{1}{\|x_i\|} \sum_{j=1}^{d} x_{ij} |j\rangle$ and $\|X\|_F = \sqrt{\sum_{i=1}^{N} \|x_i\|^2}$ is the matrix Frobenius norm.

There are multiple proposals for preparing the states given in Eq 2 and Eq 3 that use an efficient data loading structure. Here we showcase a time-efficient (poly-logarithmic in the size of the input) method of such state preparation using the quantum-accessible-data structure called the KP-tree [33]. In Appendix B we also show an alternate method of preparing such states when one is given access to the oracular quantum-random-access memory structure, whose time complexity is, in general, proportional to the square root of the size of the input unless in special cases.

5.1.2 Amplitude encoding with KP-tree

As highlighted by the original authors [33] and further in [14], the KP-tree data structure can be seen as a quantum-accessible-data structure because it is a classical tree-like data structure, which is stored in a quantum read-only-memory and accessed via superposition. It facilitates the creation of amplitude encoding states as stated in the lemmas below.

Lemma 5.1. *Let $X \in \mathbb{R}^{N\times d}$ be a given dataset. Then there exists a classical data structure to store the rows of X with the memory and time requirement to create the data structure being $T_{kp} = \mathcal{O}(Nd\log^2(Nd))$ such that, there is a quantum algorithm with access to the data structure which can perform the following unitaries (and also in superposition)*

$$|i\rangle|0\rangle \to |i\rangle \frac{1}{\|x_i\|}\sum_{j=1}^{d} x_{ij}|j\rangle \tag{4}$$

$$|0\rangle \to \frac{1}{\|X\|_F}\sum_{i=1}^{N}\|x_i\||i\rangle \tag{5}$$

in time $T = \mathcal{O}(poly\log(Nd))$.

Proof. We provide proof of this lemma in Appendix C. □

Similarly, using the results of Lemma 5.1, we also have the following two Lemmas to load the elements of the columns of the matrix X which correspond to the feature vectors, and also to load the elements of the label vector Y.

Lemma 5.2 (Superposition over example columns)**.** *Let $X \in \mathbb{R}^{N\times d}$ be a given dataset. Then there exists a classical data structure to store the columns of X given by feature vectors $x^{(j)} := (x_{1j}, \cdots, x_{Nj}), j \in [d]$ with the memory and time requirement to create the data structure being $\mathcal{O}(Nd\log^2(Nd))$ such that, there is a quantum algorithm with access to the data structure which can perform the following unitary (and also in superposition)*

$$|0\rangle|j\rangle \to \left(\frac{1}{\|x^{(j)}\|}\sum_{i=1}^{N} x_{ij}|i\rangle\right)|j\rangle = \left|x^{(j)}\right\rangle|j\rangle \tag{6}$$

in time $T = \mathcal{O}(poly\log(Nd))$.

Lemma 5.3 (Superposition over label data)**.** *Let $Y = [y_1, \cdots, y_N]^T \in \mathbb{R}^{N\times 1}$ be a given label vector. Then there exists a classical data structure to store the elements of Y with the memory and time requirement to create the data structure being $\mathcal{O}(N\log^2(N))$ such that, there is a quantum algorithm with access to the data structure which can perform the following unitaries (and also in superposition) in time $T = \mathcal{O}(poly\log(N))$ which can perform the following operation*

$$|0\rangle \to \frac{1}{\|Y\|}\sum_{i=1}^{N} y_i|i\rangle, \tag{7}$$

where $\|Y\| = \sqrt{\sum_{i=1}^{N} y_i^2}$.

5.2 Feature weight estimation

Statistical analysis provides various quantitative techniques to assess the relationship between two variables [2]. When dealing with two numerical variables, a common method to quantify their bivariate linear correlation, indicating the predictability of one variable based on the other, is through the Pearson correlation coefficient [48]. In our context, we are interested in examining the relationship between each feature vector and the target label vector using the Pearson correlation coefficient. This approach applies to regression tasks where both the feature vector and label vector consist of numerical variables.

For binary classification tasks, point-biserial correlation measures the connection between numerical feature vectors and a categorical label vector with two classes (typically denoted as $\{0, 1\}$) [37]. It is worth noting that point-biserial correlation essentially represents a specialized application of the Pearson correlation for scenarios involving binary label vectors. Therefore, we choose to employ the Pearson correlation to quantify feature weights for both regression and binary classification tasks.

5.2.1 Pearson correlation coefficient

The Pearson correlation, also known as the Pearson product-moment correlation coefficient, provides a normalized measure of the covariance between two data sets. In essence, it quantifies the relationship between two variables by comparing the ratio of their covariance to their individual standard deviations. While the Pearson correlation effectively captures linear relationships between variables, it may not accurately represent non-linear associations. Pearson correlation assumes normal distribution for both variables and can be applied to assess relationships between either nominal or continuous variables.

Our objective is to compute the Pearson correlation between each feature vector $x^{(j)}$ and the label vector Y to ascertain the importance of each feature in the prediction. Consider we want to compute the correlation between the feature j with entries $x^{(j)} = \{x_{1j}, \cdots x_{Nj}\}$ and the output label Y with entries $y_1, \cdots, y_N$. The Pearson correlation coefficient between the two N dimension

vectors is defined as

$$w_j = \frac{\sum_{i=1}^{N}(x_{ij}-\mu_j)(y_i-\mu_y)}{\sqrt{\sum_{i=1}^{N}(x_{ij}-\mu_j)^2}\sqrt{\sum_{i=1}^{N}(y_i-\mu_y)^2}} \forall j \in [d], \tag{8}$$

where $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$ and $\mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i$.

With the aforementioned amplitude-encoded states, we propose two efficient (in N) methods of quantumly computing the Pearson correlation coefficient compared to the classical method [**?**] that takes time $\mathcal{O}(N)$, whereas the proposed quantum algorithm takes time poly-logarithmic in N when the data is encoded in amplitude-encoded states. Whereas the first method has a complexity that scales on the error as $\mathcal{O}(1/\epsilon^2)$, the second method improves the error dependence quadratically to $\mathcal{O}(1/\epsilon)$. Here the error dependence is defined by $|\bar{w_j} - w_j| \leq \epsilon$, where $\bar{w_j}$ is the estimated correlation. We note that while we consider amplitude encoding states in the following methods, the feature and label vectors can also be encoded in QRAM-like states (e.g., $\sum_i^N |i\rangle |y_i\rangle$) and we can use quantum counting based algorithms [59] to efficiently calculate the correlation coefficient (see Appendix B.1 for details).

5.2.2 Method 1: Inverse error-squared dependence on run time

Theorem 5.1. *Given access to the amplitude-encoded states for feature vectors $|x^{(j)}\rangle, j \in [d]$ and the label vector $|Y\rangle$ along with their norms $\|x^{(j)}\|$, $\|Y\|$ which are prepared in time $T = \mathcal{O}(poly\log(Nd))$, there exists a quantum algorithm to estimate the Pearson correlation coefficients $\bar{w_j}, \forall j \in [d]$ in time $\mathcal{O}(\frac{Td\eta}{\epsilon^2})$, where $|\bar{w_j} - w_j| \leq \epsilon$, and $\eta = \frac{7 \cdot max\left(\|x^{(j)}\|\|Y\|, \|x^{(j)}\|^2, \|Y\|^2\right)}{N \cdot min\left(\sigma_{x^{(j)}}\sigma_Y, \sigma^2_{x^{(j)}}, \sigma^2_Y\right)}$, where $\sigma_{x^{(j)}}$ and σ_Y denote the standard deviation for $x^{(j)}$ and Y.*

Proof. We provide the proof of Theorem 5.1 in Appendix D. □

5.2.3 Method 2: Inverse error dependence on run time

Theorem 5.2. *Given access to the amplitude-encoded states for feature vectors $|x^{(j)}\rangle, j \in [d]$ and the label vector $|Y\rangle$ along with their norms $\|x^{(j)}\|$, $\|Y\|$ which are prepared in time $T = \mathcal{O}(poly\log(Nd))$, there exists a quantum algorithm to estimate each Pearson correlation coefficient $\bar{w_j}, j \in [d]$ in time $T_w = \mathcal{O}(\frac{Td\eta}{\epsilon})$, where $|\bar{w_j} - w_j| \leq \epsilon$, and $\eta = \frac{7 \cdot max\left(\|x^{(j)}\|\|Y\|, \|x^{(j)}\|^2, \|Y\|^2\right)}{N \cdot min\left(\sigma_{x^{(j)}}\sigma_Y, \sigma^2_{x^{(j)}}, \sigma^2_Y\right)}$, where $\sigma_{x^{(j)}}$ and σ_Y denote the standard deviation for $x^{(j)}$ and Y.*

Proof. We provide the proof of Theorem 5.2 in Appendix E. The key idea is to augment the proof technique in Theorem 5.1 with the amplitude estimation step Appendix A.1 to provide a quadratic improvement in the error dependence. □

5.2.4 Normalizing the feature weights

Once we obtain the feature weights $w = [w_1, \cdots, w_d]^T$, we take the absolute values of the weights since the absolute value of the Pearson correlation coefficient is the true indicator of the strength of the relationship between the two variables. Subsequently, in order to obtain the relative impact of each feature, we normalize the feature weights resulting in the normalized feature weights as

$$w_{\text{norm}} = \frac{1}{\|w\|}[|w_1|, \cdots, |w_d|]^T, \tag{9}$$

where $\|w\| = \sqrt{\sum_{j=1}^{d} w_l^2}$. This makes the norm of $\|w_{\text{norm}}\| = 1$. For simplicity, we refer to w_{norm} as w henceforth.

5.3 Clustering: supervised q-means to expand the decision tree

After having computed the feature weights $w = [w_1, \cdots, w_d]^T$, we proceed with clustering on the parent node to generate k children nodes. This process is repeated sequentially until the maximum tree depth D is reached. To perform this clustering, we leverage the unsupervised q-means algorithm developed by Kerenidis *et al.* [31] to include the feature weights in the distance estimation in such a way that the training examples are assigned to the centroid labels according to the minimum weighted distances defined in Eq 1. q-means is analogous to the classical robust k-means clustering, also known as δ-k-means, where the idea is to start with k initial centroids which are either chosen randomly or using heuristics like

k-means++ [4]. The algorithm then alternates between two steps: (a) each data point is assigned to the closest centroid thus forming total k disjoint clusters, and (b) the centroid vectors are updated to the average of data points assigned to the corresponding cluster. The two steps are repeated for a total of K times to find the approximate *optimal* centroid points. We make the same assumption of working with datasets that allow for good clustering as made by the Kerenidis *et al.* [31]. A dataset being *well-clusterable* means that the k clusters arise from picking k well-separated centroids, and then each point in the cluster is sampled from a Gaussian distribution with small variance centered on the centroid of the cluster. This implies that the size of each cluster $\mathcal{C}_j$ can be bounded by $|\mathcal{C}_j| = \Omega(\frac{N}{k})$. For Des-q the assumption is that the data is well-clusterable at root (i.e., considering the dataset entirely) and at each internal node of the tree, which contains a subset of this dataset. For such well-clusterable datasets we can obtain a poly-logarithmic time complexity in terms of number of samples N. Moreover, in order to bound the norms of X and Y, we make the assumption that they have bounded constant entries allowing to take $\|x^{(l)}\| = \mathcal{O}(\|x^{(l)}\|_\infty\sqrt{N}) = \mathcal{O}(\sqrt{N})$ and $\|Y\| = \mathcal{O}(\|Y\|_\infty\sqrt{N}) = \mathcal{O}(\sqrt{N})$, which we will use to bound some complexities in Theorem 5.6 and 5.7.

Our main result for performing supervised clustering for any given parent node is the following

Theorem 5.3. *Given quantum access to the dataset X in time $T = \mathcal{O}(poly\log(Nd))$, the supervised q-means algorithms takes K iterations steps to output with a high probability the centroids $\overline{c_1}, \cdots, \overline{c_k}$ that are arbitrarily close to the output centroids of the δ-k-means algorithm $c_1, \cdots, c_k$ in time complexity*

$$\mathcal{O}\left(\text{poly}\log(Nd)\frac{Kk^{\frac{5}{2}}d\log(k)\log(1/\Delta)\eta_1}{\epsilon_1\epsilon_2}\right), \quad (10)$$

where $\eta_1 = max_i(\|x_i\|^2)$ and ϵ_1 is the error in the estimation of the weighted inner product between each sample and centroid such that $|\overline{I_w(x_i, c_j)} - I_w(x_i, c_j)| \leq \epsilon_1$ with a probability at least $1 - 2\Delta$ (refer to Theorem 5.4) and ϵ_2 is the error in the centroid estimation as $\|\overline{c_j} - c_j\|_\infty \leq \epsilon_2\ \forall j \in [k]$ (refer to Theorem 5.6).

Here, we present an overview of the key steps in our proposed quantum-supervised algorithm that utilizes the q-means algorithm [31]. Initially, quantum subroutines are employed to assign training examples to their nearest centroids using weighted distance estimation. Subsequently, a quantum centroid update subroutine calculates the new centroid values. The primary subroutines, which we will elaborate on in the following subsections, are as follows.

1. *Weighted centroid distance estimation.*
2. *Cluster assignment.*
3. *Centroid states creation.*
4. *Centroid update.*

5.3.1 Weighted centroid distance estimation

The algorithm starts by selecting k d-dimensional vectors as initial centroids $c_1^0, \cdots, c_k^0$ (for ease of notation, we refer to them as $c_1, \cdots, c_k$) which can be chosen either randomly or by an efficient procedure such as k-means++ [4]. Next, one performs the Hadamard multiplication (also called element-wise multiplication) between each centroid vector and the weight vector w resulting in the *weighted*-centroids as

$$c_{j,w} = w \circ c_j = [w_1 c_{j1}, \cdots w_d c_{jd}]^T \quad j \in [k]. \quad (11)$$

This operation takes a total time $\mathcal{O}(kd)$. Next, the k weighted centroids are loaded in KP-tree (Sec 5.1.2) in time $T_{kp}^c = \mathcal{O}(kd\log^2(kd))$ and retrieved as amplitude-encoded states in time $\mathcal{O}(poly\log(kd))$ by doing

$$|j\rangle|0\rangle \to |j\rangle|c_{j,w}\rangle = |j\rangle \frac{1}{\|c_{j,w}\|}\sum_{l=1}^{d} w_l c_{jl}|l\rangle, \quad (12)$$

where $\|c_{j,w}\| = \sqrt{\sum_{l=1}^{d} w_l^2 c_{jl}^2}$ is the norm of the weighted centroid.

Further, using Lemma 5.1, we can also query amplitude-encoded training example states $|x_i\rangle$ in time $T = \mathcal{O}(poly\log(Nd))$ as

$$|i\rangle|0\rangle \to |i\rangle|x_i\rangle, \quad (13)$$

where $|x_i\rangle = \frac{1}{\|x_i\|}\sum_{j=1}^{d} x_{ij}|j\rangle$.

The idea of assigning all the N training examples to their closest centroid begins by estimating the weighted distance between the examples (stored in quantum superposition) and each

of the k centroid vectors (also stored in quantum amplitude encodings) and then selecting the minimum distance between the example and the k available centroids. Here we assign the examples based on them having the maximum *weighted* inner product, or overlap, with respect to the k centroid vectors. The weighted inner product between an example x_i and the centroid c_j is defined as

$$I_w(x_i, c_j) = \sum_{l=1}^{d} w_l x_{il} c_{jl} = x_i \cdot c_{j,w} = I(x_i, c_{j,w}). \tag{14}$$

Let us first look at the procedure of estimating the inner product between all the examples and the centroids.

Theorem 5.4. *Given quantum access to the dataset X in time $T = \mathcal{O}(poly\log(Nd))$ and quantum access to the weighted centroids $c_{1,w}, \cdots, c_{k,w}$ in time $\mathcal{O}(poly\log(kd))$, then for any $\Delta > 0$ and $\epsilon_1 > 0$, there exists a quantum algorithm such that*

$$\begin{aligned} &\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle \otimes_{j\in[k]} (|j\rangle |0\rangle) \\ &\rightarrow \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle \otimes_{j\in[k]} (|j\rangle \left|\overline{I_w(x_i, c_j)}\right\rangle), \end{aligned} \tag{15}$$

where $I_w(.)$ is the weighted inner product between the two vectors and $|\overline{I_w(x_i, c_j)} - I_w(x_i, c_j)| \leq \epsilon_1$ with probability at least $1 - 2\Delta$, in time $T_{cd} = \mathcal{O}(Tk\frac{\eta_1}{\epsilon_1}\log(1/\Delta))$, where $\eta_1 = max_i(\|x_i\|^2)$.

Proof. We prove the above Theorem 5.4 in Appendix F. □

5.3.2 Cluster assignment

Once we have the superposition state of Theorem 5.4, we can use the following theorem to assign each training example x_i to the closest centroid label referred by *centroid-label*$_i \in [k]$[6]. In our case, this would correspond to the largest weighted inner product between each example and the k centroid points.

[6]Note that the centroid label is different from the original dataset label. Since we are doing k-way clustering, thus each cluster centroid c_j is assigned a label j, for $j \in [k]$

Theorem 5.5 (Finding maximum weighted inner product)**.** *Given k different $\log p$ size registers $\otimes_{j\in[k]}|\overline{I_w(x_i, c_j)}\rangle$ whose creation time is $T_{cd} = \mathcal{O}(Tk\frac{\eta_1}{\epsilon_1}\log(1/\Delta))$, where T, ϵ_1 and Δ are defined in Theorem 5.4, there is a quantum map which performs the following operations*

$$\begin{aligned} &\otimes_{j\in[k]} |\overline{I_w(x_i, c_j)}\rangle |1\rangle \\ &\rightarrow \otimes_{j\in[k]}|\overline{I_w(x_i, c_j)}\rangle|arg\text{-}max(\overline{I_w(x_i, c_j)})\rangle \end{aligned} \tag{16}$$

incurring a total time $T_l = \mathcal{O}(k\log p) + T_{cd} = \mathcal{O}(k\log p + Tk\frac{\eta_1}{\epsilon_1}\log(1/\Delta)) = \mathcal{O}(Tk\frac{\eta_1}{\epsilon_1}\log(1/\Delta)) = T_{cd}$. Note that we removed the first term as $\log p$ is a constant as it is the size of the registers to encode each inner product $|\overline{I_w(x_i, c_j)}\rangle$ (Theorem 5.4).

Proof. This process can be done by having an additional register initialized in $|1\rangle$. Then for $2 \leq j \leq k$, a total of k repeated comparisons are performed of the two inner product registers, i.e., if the value of the inner product register corresponding to centroid c_j is greater than the value corresponding to c_{j+1}, then the last register takes the centroid label value $j+1$ and so on. Thus the total time complexity of this procedure is $\mathcal{O}(k\log p)$. This allows us to produce the state

$$\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle |centroid\text{-}label_i\rangle, \tag{17}$$

where *centroid-label*$_i$ is the centroid label corresponding to the maximum weighted inner product with respect to the k centroids for the example x_i. Thus the total time complexity (including the previous step of weighted inner product estimation) is T_l. □

5.3.3 Updated centroid vectors creation

Once the clusters are assigned, the next step is to perform the averaging of the values in the cluster (i.e., an average of weighted examples with the same centroid label) to obtain the new cluster centroids. One way to do this is to note that the state in Eq 17 can be re-written as

$$\sum_{j=1}^{k} \sqrt{\frac{|\mathcal{C}_j|}{N}} \left(\frac{1}{\sqrt{|\mathcal{C}_j|}} \sum_{i\in\mathcal{C}_j} |i\rangle\right) |j\rangle = \sum_{j=1}^{k} \sqrt{\frac{|\mathcal{C}_j|}{N}} |\xi_j\rangle |j\rangle, \tag{18}$$

where $|\xi_j\rangle$ corresponds to the uniform superposition over the indices in the cluster $\mathcal{C}_j$ also

referred to as the characteristic vector state corresponding to the centroid label value j. As mentioned, we assume well-clusterable datasets, implying that the clusters are all non-vanishing and they have the size $|\mathcal{C}_j| = \Omega(N/k)$, we see that with $\mathcal{O}(k \log k)$ measurements of the final register (using coupon collector arguments [7]), we obtain all the k index states $|\xi_j\rangle, j \in [k]$ with $\Omega(1)$ probability. Thus we can obtain each characteristic vector state $|\xi_j\rangle$, $j \in [k]$ in time $T_\xi = \mathcal{O}(T_l \log k)$ such that the total time to obtain all the characteristic vectors is kT_ξ.

Now, we use the following theorem to obtain all the k updated classical centroid vectors denoted by $c_1^1, \cdots, c_k^1$ where $c_j^1 \in \mathbb{R}^d \ \forall j \in [k]$ and the superscript denotes the updated centroid vectors at the cluster iteration 1. For ease of notation, we omit the superscript and denote them as $c_1, \cdots, c_k$.

Theorem 5.6. *Given access to the characteristic vector states $|\xi_j\rangle \ \forall j \in [k]$ where each state is prepared in time $T_\xi = \mathcal{O}(T_l \log k)$ and the amplitude-encoded states for feature vectors $|x^{(l)}\rangle \ \forall l \in [d]$ which can be prepared in time $T = \mathcal{O}(poly\log(Nd))$, there exists a quantum algorithm to obtain the updated centroid vectors $\overline{c_1}, \cdots, \overline{c_k}$ such that $\|\overline{c_j} - c_j\|_\infty \leq \epsilon_2 \ \forall j \in [k]$ in time*

$$T_{sup\text{-}cluster} = \mathcal{O}\left(\frac{T_\xi k^{\frac{3}{2}} d}{\epsilon_2}\right),$$

where $c_j = X^T \xi_j$ is the true mean of the weighted examples in the cluster and thus the true updated centroid vector of c_j.

Proof. We prove the Theorem 5.6 in Appendix G. □

We can now easily compute the time taken to generate the first iteration of centroid vectors[7]. The total time taken is the combination of time to load the initial centroid vectors in KP-tree and to perform supervised clustering (a combination of data update and clustering). This can be written as

$$\begin{aligned}
T_{\text{iter-1}} &= T_{kp}^c + T_{\text{sup-cluster}} \\
&= \mathcal{O}(\log^2(kd) \cdot kd) + \mathcal{O}\left(T_\xi \frac{k^{\frac{3}{2}}}{d} \epsilon_2\right) \\
&= \mathcal{O}(\log^2(kd) \cdot kd) + \mathcal{O}\left(T_l \frac{k^{\frac{3}{2}}}{d} \log(k) \epsilon_2\right) \\
&= \mathcal{O}(\log^2(kd) \cdot kd) + \mathcal{O}\left(T_{cd} \frac{k^{\frac{3}{2}}}{d} \log(k) \epsilon_2\right) \\
&= \mathcal{O}(\log^2(kd) \cdot kd) + \mathcal{O}\left(poly\log(Nd) \log(1/\Delta) \eta_1 \frac{k^{\frac{5}{2}}}{d} \log(k) \epsilon_1 \epsilon_2\right) \\
&= \mathcal{O}\left(\frac{poly\log(Nd) \log(1/\Delta) \eta_1 k^{\frac{5}{2}}}{d} \log(k) \epsilon_1 \epsilon_2\right),
\end{aligned} \tag{19}$$

where η_1 and ϵ_1 are defined in Theorem 5.4) and ϵ_2 in Theorem 5.6). Further, T_{kp}^c is the time to load the weighted centroid vector in the KP-tree as mentioned in Sec 5.3.1.

[7]For the time being we will exclude the time taken to load the initial data X, X^T, Y in the KP-tree and the time to compute the Pearson correlation weight vector. This is because these are one-time processes and they will be included in the total decision tree construction time in the end.

5.3.4 Repeating clustering iteration step

Once we have obtained the k classical centroid vectors for one iteration, we repeat the above 3-step process K times to obtain the final centroid vectors at the K-th iteration. This creates the first depth of the decision tree with a total of k children nodes corresponding to the k clusters generation. The total time complexity of this procedure is (excluding the time to load the initial data X, X^T, Y into the KP-tree and the time to

compute the Pearson correlation weight vector)

$$T_{\text{depth:}[0\to1]} = KT_{\text{iter-1}}. \tag{20}$$

5.4 Tree growth

The previous steps in Sec 5.2-5.3 show how to grow the tree from the root node to the k children nodes at depth 1. As a result we obtain the k centroid vectors at depth 1 $\{c_1^1, \cdots, c_k^1\}$ where the superscript denotes the centroids at tree depth 1. Subsequently, in order to expand the tree further, we need to perform clustering again on each of the k children nodes. This is repeated till we reach the final tree depth D. The challenge here is that we started with all the samples in superposition at root ($\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|i\rangle|x_i\rangle$). After having performed clustering, now we require the superposition of the samples at each children node to continue expanding the tree. This state is the index superposition state. For the j-th cluster at depth l ($\mathcal{C}_{j,l}$), it is defined as $|\xi_j\rangle = \frac{1}{\sqrt{|\mathcal{C}_{j,l}|}}\sum_{i\in\mathcal{C}_{j,l}}|i\rangle$. Creating this state is non-trivial, as we will show in Section 5.5.

Let us see how to further expand the tree by expanding at the nodes from depth l to $l+1$. This can be done in the following two sequential steps,

1. Let us denote the node j in depth l with the cluster $\mathcal{C}_{j,l} \forall j \in [k^l]$ and $l \in [1, D-1]$. The first step is to create the superposition over the examples in the cluster $\mathcal{C}_{j,l}$.

2. Perform the clustering to create k children nodes for each parent cluster node $\mathcal{C}_{j,l}$ at depth-l.

For 1, we create the superposition over the examples in node $j \in [k^l]$. Let us denote the nodes by their centroids vectors $c_j^l, j \in [k^l]$ where the superscript denotes the depth of the tree and the subscript denotes the node. Next, we perform the Hadamard multiplication (also called element-wise multiplication) between each centroid vector and the weight vector w resulting in the *weighted*-centroids

$$c_{j,w} = w \circ c_j = [w_1 c_{j1}, \cdots w_d c_{jd}]^T, \;\; j \in [k^l]. \tag{21}$$

This operation takes a total time $\mathcal{O}(k^l d)$. The k^l weighted centroids are then loaded in KP-tree (Sec 5.1.2) in time $T_{kp}^c = \mathcal{O}(k^l d \log^2(kd))$ and retrieved as amplitude-encoded states in time $\mathcal{O}(poly\log\left(k^l d\right))$.

For 2, we perform clustering for each node at depth-l. For this, similar to Sec 5.3.1, the idea is to create the superposition state of the indices of N examples in the dataset and their weighted inner product distance with the k^l available centroid vectors. For simplicity, again let us refer to $\{c_j^l\}$ as $\{c_j\}$. We utilize Theorem 5.4 but in this case for the k^l nodes. Therefore, the complexity time is $T_{cd}^{k^l} = \mathcal{O}(Tk^l \cdot \frac{\eta_1}{\epsilon_1}\log(1/\Delta))$, where we utilize the super-index k^l to refer to the dependency on the k^l nodes.

Next, we can coherently find the maximum weighted inner product for each example state and the k^l centroid vectors. This allows one to cluster the original data into the disjoint clusters at depth l. This is done using Theorem 5.5 for k^l nodes. In this case, the complexity time is $T_l^{k^l} = \mathcal{O}(T_{cd}^{k^l} + k^l \log p) = T_{cd}^{k^l}$ due to the same argument as before.

The resulting quantum state is

$$\sum_{j=1}^{k^l}\sqrt{\frac{|\mathcal{C}_{j,l}|}{N}}\left(\frac{1}{\sqrt{|\mathcal{C}_{j,l}|}}\sum_{i\in\mathcal{C}_{j,l}}|i\rangle\right)|j\rangle = \sum_{j=1}^{k^l}\sqrt{\frac{|\mathcal{C}_{j,l}|}{N}}|\xi_j\rangle|j\rangle. \tag{22}$$

Upon measuring the last register of the above superposition state $\mathcal{O}(k^l \log k^l)$ times, we obtain the cluster index states $|\xi_j\rangle, \forall j \in [k^l]$ with an $\Omega(1)$ probability. Thus each state $|\xi_j\rangle$ can be prepared in time

$$T_\xi^{k^l} = \mathcal{O}(T_l^{k^l} \cdot \log k^l) = \mathcal{O}(T_l^{k^l} \cdot l \log k). \tag{23}$$

Upon querying the KP-tree in Theorem 5.1 separately with each index state $|\xi_j\rangle$, we obtain the state

$$|\xi_j\rangle|0\rangle \to \frac{1}{\sqrt{|\mathcal{C}_{j,l}|}}\sum_{i\in\mathcal{C}_{j,l}}|i\rangle|x_i\rangle \;\; \forall j \in [k^l], \tag{24}$$

where $|x_i\rangle = \frac{1}{\|x_i\|}\sum_{j=1}^{d}x_{ij}|j\rangle$ in time $T_{\text{cs}}^{k^l} = \mathcal{O}(T_\xi^{k^l} poly\log(Nd))$. This then allows us to create the superposition of the example states in the node j at depth l.

Considering steps 1 and 2, the total time to expand all nodes at depth-l to get the k^{l+1} children nodes is the time $T_{\text{cs}}^{k^l}$ to generate the superposition states for each cluster multiplied by the time to do the clustering of each node, which is the same as clustering the root node (i.e., clustering from the root to depth 1 nodes $T_{\text{depth:}[0\to1]}$). Therefore, the total complexity time is

$$
\begin{aligned}
T_{\text{depth:}[l\to l+1]} &= \mathcal{O}(k^l d + k^l \cdot T_{\text{cs}}^{k^l} \cdot T_{\text{depth:}[0\to 1]}) \\
&= \mathcal{O}(k^l d + k^l \cdot T_{\text{cd}}^{k^l} \cdot l \cdot \log k \cdot poly\log(Nd) \cdot KT_{\text{iter-1}}) \\
&= \mathcal{O}\left(poly\log(Nd)\frac{Klk^{2(l+\frac{5}{4})}d\log^2(k)\log^2(1/\Delta)\eta_1^2}{\epsilon_1^2\epsilon_2}\right).
\end{aligned} \tag{25}
$$

The tree keeps growing using the previous strategy until it reaches the maximum allowed depth D or until a stopping criterion is achieved. The total time taken for the algorithm to reach from the root node to the last node at depth D (excluding the time to load the initial data X, X^T, Y into the KP-tree and the time to compute the Pearson correlation weight vector at the root node) is

$$
\begin{aligned}
T_{\text{depth:}[0\to D]} &= T_{\text{depth:}[0\to 1]} + \sum_{l=1}^{D-1} T_{\text{depth:}[l\to l+1]} \\
&= \mathcal{O}\left(poly\log(Nd)\frac{KDk^{2D+\frac{1}{2}}d\log^2(k)\log^2(1/\Delta)\eta_1^2}{\epsilon_1^2\epsilon_2}\right).
\end{aligned} \tag{26}
$$

5.5 Leaf label assignment

Once the tree reaches the final depth D, it consists of k^D leaf nodes. Now our objective is to compute the label values for each of the leaf nodes, which are clusters $\mathcal{C}_{j,D}, j \in [k^D]$ containing some training samples. For the task of regression, the label value is simply the mean of the label values of these samples i.e.,

$$
label_j = \frac{1}{|\mathcal{C}_{j,D}|}\sum_{i\in\mathcal{C}_{j,D}} y_i. \tag{27}
$$

For the task of binary regression with label set $\mathcal{M} \in \{0, 1\}$, the leaf label value can similarly be computed by computing the mean of the label values of the examples in the cluster. If the mean is less than 0.5, it is assigned the value 0, and 1 otherwise.

The mean value for any leaf node can be calculated by first creating a superposition of the indices in the cluster corresponding to the leaf node, i.e., by creating the index superposition state $|\xi_j\rangle = \frac{1}{\sqrt{|\mathcal{C}_{j,D}|}}\sum_{i\in\mathcal{C}_{j,D}}|i\rangle$. Subsequently, the mean value is simply obtained from the inner product between the index superposition state and label superposition state $|Y\rangle = \frac{1}{\|Y\|}\sum_{i=1}^{N} y_i|i\rangle$ as

$$
\begin{aligned}
I(|\xi_j\rangle, |Y\rangle) &= \langle\xi_j|Y\rangle \\
&= \frac{1}{\sqrt{|\mathcal{C}_{j,D}|}\|Y\|}\sum_{i\in\mathcal{C}_{j,D}} y_i \\
&= \frac{\sqrt{|\mathcal{C}_{j,D}|}}{\|Y\|} label_j.
\end{aligned} \tag{28}
$$

We highlight the steps in greater detail below.

5.5.1 Creating index superposition states

The first step of obtaining the leaf node label is to create the superposition over the indices of the cluster $\mathcal{C}_{j,D}$ corresponding to the leaf node j i.e., our objective is to first create the state $|\xi_j\rangle$ by performing the mapping in Eq. 22 in time T_ξ as shown in Eq 23 for $l = D$ as

$$
\begin{aligned}
T_\xi &= \mathcal{O}(T_l^{k^D} \cdot D\log k) \\
&= \mathcal{O}\left(poly\log(Nd)\frac{Dk^D\log(k)\log(1/\Delta)\eta_1}{\epsilon_1}\right).
\end{aligned} \tag{29}
$$

5.5.2 Obtaining mean label value

Our next step is to query the label superposition state $|Y\rangle$ using Lemma 5.3 by doing

$$
|0\rangle \to \frac{1}{\|Y\|}\sum_{i=1}^{N} y_i|i\rangle = |Y\rangle. \tag{30}
$$

This takes time $\mathcal{O}(poly\log(N))$. Next, using the following theorem, we can obtain the mean label value for each leaf node $j \in [k^D]$.

Theorem 5.7. *Given access to the characteristic vector states $|\xi_j\rangle$ $\forall j \in [k^D]$ where each state is prepared in time $T_\xi = \mathcal{O}(T_l^{k^D} D \log k)$ and the amplitude-encoded states label superposition state $|Y\rangle$ which is be prepared in time $\mathcal{O}(poly\log(N))$, there exists a quantum algorithm to obtain the mean label values $\{\overline{label_1}, \cdots, \overline{label_{k^D}}\}$ such that $|\overline{label_j} - label_j| \leq \epsilon_3$, $\forall j \in [k^D]$ in time*

$$T_{leaf\text{-}label} = \mathcal{O}\left(\frac{T_\xi k^{\frac{3D}{2}}}{\epsilon_3}\right),$$

where $label_j$ is the true label mean of the examples in the cluster as given in Eq 27.

Proof. We prove the Theorem 5.7 in Appendix H. □

5.6 Time complexity for decision tree construction

After showcasing the method to grow the tree and compute the leaf labels, we are now in a position to calculate the total time it takes to build the decision tree. We note that our algorithm has a dependency poly-logarithmic in N. However, given the classical dataset, the total time taken to build the decision tree is the time to load the data X into the KP-tree and Y in the classical data structure, plus the time to compute the Pearson correlation coefficient weights plus the rest of the algorithmic steps.

The time taken to load the classical data in the quantum-accessible-data structure (either KP tree for X or list type data structure for Y) is

$$\begin{aligned} T_{\text{load}} &= T_{kp} + T_Y \\ &= \mathcal{O}(Nd\log^2(Nd) + N\log^2(N)) \\ &= \mathcal{O}(Nd\log^2(Nd)), \end{aligned} \tag{31}$$

where T_{kp}, T_Y are defined in Lemmas 5.1,5.3 respectively.

Next, the time taken to compute the Pearson correlation coefficients is

$$T_{\text{corr}} = T_w = \mathcal{O}\left(poly\log(Nd)\frac{d\eta}{\epsilon}\right), \tag{32}$$

where T_w is defined in Eq 90.

Therefore, performing supervised clustering consecutively until tree depth D and the leaf label assignment takes time

$$\begin{aligned} T_{\text{algo}} &= T_{\text{depth:}[0\to D]} + T_{\text{leaf-label}} \\ &= \mathcal{O}\left(poly\log(Nd)\frac{KdDk^{2D+\frac{1}{2}}\log^2(k)\log^2(1/\Delta)\eta_1^2}{\epsilon_1^2\epsilon_2}\right), \end{aligned} \tag{33}$$

where $T_{\text{depth:}[0\to D]}$ and $T_{\text{leaf-label}}$ are defined in Eq 26 and 121 respectively.

Now, we can combine the three steps to compute the total time taken for the decision tree as

$$T_{\text{des-tree}} = T_{\text{load}} + T_{\text{corr}} + T_{\text{algo}}. \tag{34}$$

We see that T_{corr} and T_{algo} have the runtime depending only poly-logarithmically in the number of examples N. However, since the initial data loading process in the suitable quantum-accessible-data structures takes time T_{load} which depends polynomially in N, this makes the initial decision tree construction algorithm to depend linearly in N.

We will see in Section 6that the decision tree retraining is extremely fast in the regime of large N due to $T_{\text{corr}} + T_{\text{algo}}$ depending only poly-logarithmically in N.

5.7 Test inference with Des-q

Once the tree is built, we need a mechanism to classify new unseen examples i.e., test unlabelled examples. Here we propose to do this classically based on the following theorem.

Theorem 5.8. *Given a test example, $x \in \mathbb{R}^d$, there exists a classical algorithm to predict the target label of the example from our proposed decision tree in time complexity*

$$T_{inf} = \mathcal{O}(kDd). \tag{35}$$

Proof. Our decision tree relies on the inner product-based clustering method (as described in

Sec 5.3). For this, we break down the process of assigning a label to a test example as

1. *Initial Depth*: We start by calculating the inner product between the test example $x \in \mathbb{R}^d$ and the k centroids at the first depth of the tree. We select the centroid with the highest inner product value, indicating it is the closest centroid to x. The time complexity for this step is $\mathcal{O}(kd)$, as computing the inner product between two d-dimensional vectors classically takes $\mathcal{O}(d)$ time.

2. *Depth Expansion*: We proceed to perform inner product estimations between x and the centroids corresponding to the k children nodes of the parent node (which corresponds to the centroid with the highest inner product from the previous depth layer). Again, we select the centroid with the highest inner product. This process repeats until we reach the maximum tree depth D, which is typically a leaf node.

3. *Leaf Node*: Finally, the target label of the example x is determined by the target label associated with the leaf node that the example is assigned to.

In summary, the total time complexity for assigning a label to a test example in this tree structure is $\mathcal{O}(kDd)$. □

6 Retraining with Des-q

After the tree has been constructed and put online to classify new examples, one would want to retrain the model with new labelled d dimensional examples $N_{\text{new}} \ll N$. This helps prevent their performance degradation which is crucial to maintain the consistent model performance. The key steps to retraining the tree include

1. Load the new N_{new} examples in the same KP-tree data structure introduced in Sec 5.1.2.

2. Recompute the feature weights for $N + N_{\text{new}}$ training dataset.

3. Perform quantum-supervised clustering to grow the tree.

4. Compute the leaf labels.

Using the same techniques used to build the tree at the initial time, we showcase in the following theorem that our algorithm for tree retraining scales only poly-logarithmically with the total number of training examples.

Theorem 6.1. *Given quantum access to previous training examples $X \in \mathbb{R}^{N\times d}$ and new training examples $X_{new} \in \mathbb{R}^{N_{new}\times d}$ new examples such that $N_{new} \ll N$, there is a quantum decision tree algorithm to retrain the tree with the new examples in time*

$$T_{retrain} \approx T_{load\text{-}new} + T_{corr} + T_{algo}, \tag{36}$$

where $T_{load\text{-}new} \approx \mathcal{O}(N_{new} d \log^2(N_{new} d))$ is the time taken to load the new examples and the corresponding labels in the KP-tree. T_{corr}, T_{algo} are computed in Eq 32, 33 and both depend poly-logarithmically in the total number of examples $N_{tot} = N + N_{new} \approx N$.

Proof. We structure the proof of the above theorem in the following steps.

1. *Loading new examples*: Our Des-q algorithm for retraining would require quantum access to not just the examples $x_i, i \in [N_{\text{tot}}]$, but also the ability to to create the feature vectors $x^{(j)}, j \in [d]$ and the label vector Y over the combined dataset of size N_{tot}. To load the examples x_i in the KP-tree data structure, we create N_{new} binary trees data structure $B_i^{new}, i \in [N_{\text{new}}]$ as highlighted in Sec 5.1.2. Creation of these trees take time $\mathcal{O}(N_{\text{new}} \log^2(N_{\text{new}} d))$. Once these trees are constructed, one can query the examples of the combined dataset $x_i, i \in [N_{\text{tot}}]$ in time $\mathcal{O}(poly\log(N_{\text{tot}} d))$.

   It follows the loading of the feature values $x^{(j)}$ of the new dataset X_{new} into the existing KP-tree data structure consisting of d binary trees $B_j, j \in [d]$. These trees were initially constructed to store the d feature values of N examples in the initial tree construction as mentioned in Lemma 5.2. One can do this by *horizontally* adding the feature values of the new examples into the existing B_j i.e., by expanding each of the d trees to add more leaf nodes which are connected all the way up to the root node. Each binary tree B_j initially had N leaf

nodes, which are now appended with N_{new} new leaves such that the root node of B_j contains the norm value of the j-th feature vector of the combined dataset consisting of N_{tot} examples i.e., $\|x^{(j)}\| = \sqrt{\sum_{i=1}^{N_{\text{tot}}} x_{ij}^2}$. Modifying all the d binary trees takes time $\mathcal{O}(N_{\text{new}} d \log^2(N_{\text{new}} d))$. Once these trees are modified, one can query the feature states $x^{(j)}$ of the combined dataset in time $\mathcal{O}(poly\log(N_{\text{tot}} d))$.

Similarly, we need to modify the KP-tree structure B which stores the label values as introduced in Lemma 5.3. The tree is modified by adding N_{new} new leaf nodes such that the root of the tree stores norm value $\|Y\| = \sum_{i=1}^{N_{\text{tot}}} y_i^2$. Modifying the tree takes time $\mathcal{O}(N_{\text{new}} \log^2(N_{\text{new}}))$ and the label state $|Y\rangle$ for the combined dataset can be retrieved in time $\mathcal{O}(poly\log(N_{\text{tot}}))$.

Thus the total time to load the examples, features and the label for the dataset N_{new} into the KP-tree data structure is

$$\begin{aligned} T_{\text{load-new}} &= \mathcal{O}(N_{\text{new}} d \log^2(N_{\text{new}} d) \\ &\quad + \mathcal{O}(N_{\text{new}} \log^2(N_{\text{new}})) \\ &= \mathcal{O}(N_{\text{new}} d \log^2(N_{\text{new}} d)). \end{aligned} \tag{37}$$

2. *Feature Weights + Supervised Clustering + Leaf Label*: Once the new data has been loaded into the KP-tree data structure, it can be retrieved in time poly-logarithmic in the total number of examples N_{tot}. Subsequently, one can apply the techniques introduced in Sec 5.2, Sec 5.3, Sec 5.4, and Sec 5.5 respectively to build the decision tree and compute the leaf labels. The time to generate the feature weights for the combined dataset is

$$\begin{aligned} T_{\text{corr-new}} &= \mathcal{O}\left(\text{poly}\log(N_{\text{tot}} d)\frac{d\eta}{\epsilon}\right) \\ &= \mathcal{O}\left(\text{poly}\log(Nd)\frac{d\eta}{\epsilon}\right) \\ &= T_{\text{corr}}. \end{aligned} \tag{38}$$

And the time to do the supervised clustering and leaf label assignment is

$$\begin{aligned} T_{\text{algo-new}} &= \mathcal{O}\left(poly\log(N_{tot}d)\frac{Dk^{2D}\log(k)\log(1/\Delta)\eta_1}{\epsilon_1}\left(\frac{Kdk^{\frac{1}{2}}\log(k)\log(1/\Delta)\eta_1}{\epsilon_1\epsilon_2}\right)\right) \\ &= T_{\text{algo}}. \end{aligned} \tag{39}$$

Thus the total time taken to retrain the decision tree is

$$\begin{aligned} T_{\text{retrain}} &= T_{\text{load-new}} + T_{\text{corr-new}} + T_{\text{algo-new}} \\ &= T_{\text{load-new}} + T_{\text{corr}} + T_{\text{algo}}. \end{aligned} \tag{40}$$

From this, we see that our decision tree for retraining has a linear dependence on N_{new} due to the $T_{\text{load-new}}$ factor, and poly-logarithmic dependence on $N_{\text{tot}} \approx N$ which comes from the term $T_{\text{corr-up}} + T_{\text{algo-up}}$. Under the small batch updates criterion $N_{\text{new}} \ll N$, we see that our algorithm for tree retraining has scales poly-logarithmically in N, thus ensuring that the tree update is extremely fast. □

7 Numerical results

This section aims to demonstrate that the proposed quantum decision tree construction method, Des-q, achieves competitive performance compared to state-of-the-art axis-parallel split-based methods commonly employing impurity measures or information gain for classification tasks and variance reduction for regression tasks. In our numerical simulations, we utilize the classical version of Des-q denoted as Des-c, which leverages the k-means clustering algorithm. The performance in terms of the accuracy of predictions of k-means closely aligns with the robust δ-k-means algorithm, as demonstrated by Kerenidis *et al.* [31], under appropriate δ selection. Further, δ-k-means is a good approximation of the per-

formance of q-means algorithm. Therefore, the performance in accuracy of predictions of Des-c closely aligns with the performance of Des-q executed on quantum hardware.

As discussed in Sec 5.2, the core component of the Des-q algorithm involves quantum estimation of the point-biserial correlation for binary classification and the Pearson correlation for regression, with numerical labels. For our numerical simulations, we calculate these values classically. For more technical details about the implementation, we refer to Appendix I.

Note that we focus on benchmarking the performance in the quality of predictions and we do not include comparisons of runtime because Des-c has no speedup over classical standard methods for decision tree construction and retraining as it utilizes the classical k-means algorithm. Therefore, in contrast to the quantum algorithm, Des-c scales polynomially in N and d. The speedup in Des-q comes from leveraging q-means for construct and retrain the tree.

Limitations. The implementation of Des-q requires the use of quantum-accesible-data structures: KP tree for X and list type data structure for Y. We refer the reader to Jaques *et al.* [30] and Allcock *et al.* [3] for a discussion on implementations. The speedup of Des-q is in the asymptotic regime. Therefore, it is needed that these structures handle large datasets, which can be challenging. However, some erasing techniques may be used or compression methods (e.g., with Principal Component Analysis) over old data to allow for space for more recent data. In terms of circuit requirement, since we perform quantum clustering in a sequential manner to build the tree, it is anticipated that the resulting circuits will be quite deep, necessitating either extended coherence times or the use of error correction techniques. In terms of number of qubits, this will depend on the data size d. Although this number may be big, the proposed algorithm targets datasets where $N \gg d$.

We hope that the proposed quantum method may trigger interesting future works that provide a quantitative resource analysis of hardware requirements and potential hardware demonstrations, as the hardware keeps on developing.

7.1 Main results

We present our results using four datasets comprising numerical features: PIMA [52], Spambase [28], Blood [64], and Boston housing [24]. Among these, the first three datasets are binary classification tasks, while Boston housing has numerical label values making it suitable for the task of regression. Their characteristics are shown in Table 1 where *Instances* denotes the number of labeled examples, and *Features* denotes the number of features (d) for each example. For all the considered datasets, we split the data into train and test using a ratio of 0.3 and created ten train-test of equally sized subsets of data, called folds, for binary classification dataset and five for the regression dataset. These folds maintain the proportion of labels. Subsequently, each fold is used to train the decision tree model and the performance is evaluated in both train and test sets. We perform standard data normalization techniques on each fold, ensuring that each feature has a mean of 0 and a standard deviation of 1.

Task	Dataset	Instances	Features
Classification	PIMA	768	8
	Spambase	4601	57
	Blood	748	4
Regression	Boston housing	506	8

Table 1: Characteristics of the datasets used for the performance benchmark.

We compare the performance of Des-c for multi-way decision tree construction against the state-of-the-art decision tree construction method. In particular, we employ the DecisionTree class for classification and the DecisionTreeRegressor class for regression, with both classes being implemented in scikit-learn [49]. We set the criterion, the impurity measure, to be the entropy for classification and mean square error for regression. The only hyperparameter of this method that we modify for the experiments is tree depth. For Des-c we evaluate its performance for different number of clusters k, ranging from two to seven, depending on the dataset. We include in the data availability section below the link to a repository with the post-processed data utilized for the experiments, the results obtained and the Jupyter notebooks to reproduce the results in the article.

The main performance results of Des-c against the baseline decision tree method are highlighted in Table 2. The metric used to benchmark the performance, shown in column *Performance*, is the accuracy for the classification task and the root mean square error (RMSE) for regression, which is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}, \qquad (41)$$

where N is the number of samples, Y_i is the label of the i-th sample and $\hat{Y}_i$ is the predictor obtained for that same sample.

The results show that the performance of the proposed method, Des-c, matches the baseline across all analyzed datasets. This comparison is made at the same tree depth, affirming that Des-c competes effectively with the baseline. We also conducted evaluations with deeper trees, extending up to 5 for classification tasks and 10 for regression tasks. It becomes evident that for depths beyond 2, all models exhibit a decline in test accuracy, signifying the onset of overfitting. Furthermore, it is noteworthy that while the baseline relies on binary splits, Des-q employs multi-way splits, and the optimal results were achieved with a number of clusters (denoted as *Clusters* in the table) greater than two. Importantly, an increase in the number of clusters (k) does not introduce significant complexity to the algorithm (as indicated by Eq 33), as $k \ll N$, where N corresponds to the number of training samples. In summary, the proposed method demonstrates competitive performance when compared to state-of-the-art approaches.

In the following subsections, we provide a more detailed examination of the results for both the classification task (Sec 7.2) and regression (Sec 7.3). Not only do we compare Des-c to the baseline but also to the same method but without incorporating feature weight in order to highlight the benefits that the incorporation of these weights in the distance metric (Eq 1), part of the supervised clustering, brings about. In each subsection, we assess the performance of tree inference with test data. Furthermore, we discuss the tree construction performance, specifically how well it reduces the impurity measure (entropy for classification and variance for regression) and its accuracy with the training data. This analysis is crucial to determine whether the tree construction aligns with expectations, which involve reducing the impurity measure, despite the fact that the proposed method does not directly optimize this measure.

7.2 Classification

For this task, Des-c utilizes the point-biserial correlation as the feature weight. We compare this approach against using the same method without any feature weights, which corresponds to setting $w_l = 1$ for all $l \in [d]$ in the distance calculations within Des-c (as shown in Eq 14). We evaluate the performance of the proposed algorithm against a state-of-the-art method, which optimizes the splits to reduce entropy. Although Des-c does not directly optimize splits based on entropy, we expect to observe a reduction in entropy as the tree grows. We define the entropy of a tree trained up to depth D as the weighted sum of the entropy of its individual leaves, following this formula: $E_D = \sum_{i=1}^{n} f_i e_i$, where f_i represents the fraction of samples in the i-th node, e_i denotes the entropy at that node, and n is the total number of leaves in the tree.

7.2.1 Tree construction: performance with training data

The conclusions we will discuss regarding tree construction have been observed for the three datasets considered. To facilitate visualization and discussion, we will focus on the results for the PIMA dataset, but it is important to note that these conclusions apply across all datasets. We begin our analysis with a depth one ($D = 1$), which corresponds to the root node and the first level of children nodes. In Fig. 2, we compare the entropy ($E_{D=1}$) obtained with Des-c, incorporating feature weights (point-biserial correlation), to the entropy obtained without weights (*no weight*) as a function of the number of clusters. Additionally, we compare these results to the entropy obtained by the binary decision tree (baseline method) at depth one.

We observe an overall reduction in entropy as the number of clusters increases. As expected, the inclusion of label information for supervised clustering results in a more pronounced reduction in entropy as the tree depth increases, as illustrated in Figure 3(a). In practice, while not incorporating feature weights results in an entropy

Task	Dataset	Model	Performance	Tree size	Tree depth	Clusters
Classification	PIMA	Baseline	74.64% ± 2.81	7	2	2
		Des-c	70.34% ± 4.53	52.2	2	7
	Spambase	Baseline	81.89% ± 2.32	7	2	2
		Des-c	80.71% ± 9.38	27	2	5
	Blood	Baseline	77.07% ± 2.10	7	2	2
		Des-c	77.26% ± 2.05	12.8	2	3
Regression	Boston Housing	Baseline	0.053 ± 0.007	7	2	2
		Des-c	0.053 ± 0.007	20	2	4

Table 2: Main results of the performance in the test for two tasks: binary classification and regression and four datasets. The performance metric utilized for classification is the accuracy (in %) and the one for regression is the RMSE. The values correspond to the average and the standard deviation across the folds considered. We compare the classical implementation of the quantum proposed algorithm, referred as Des-c, to the baseline, a binary decision tree built based on the reduction of entropy for classification and mean-square error for regression. For classification, Des-c incorporates the point-biserial correlation as a feature weight whereas for regression, it utilizes the Pearson correlation. Note that beyond depth 2 models start to overfit. The results reported correspond to the minimal number of clusters for which Des-c performs on par with the baseline. The tree size is defined as the average number of tree nodes across the folds.

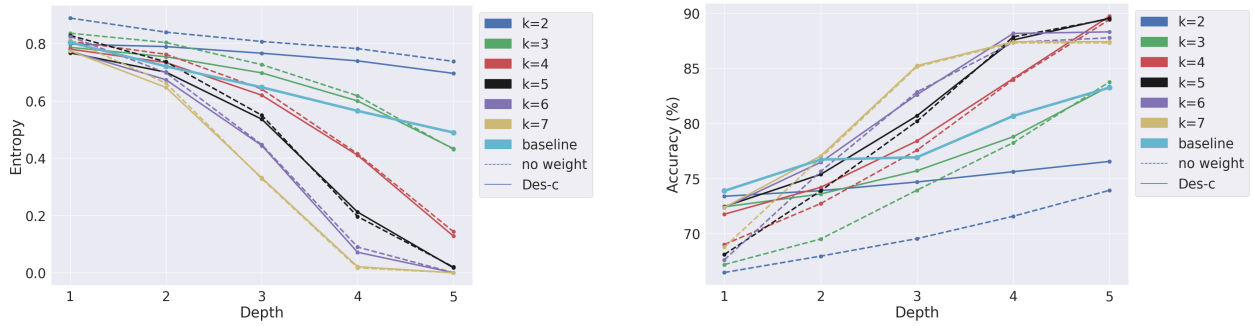
that is not significantly different from the baseline, incorporating feature weights reduces the entropy even more than the baseline. This improvement in entropy improves the performance in training accuracy where Des-c achieves competitive and even better performance compared to the baseline for some higher values of the number of clusters ($k \in [4, 5, 6, 7]$) and for higher tree depths ($d \in [3, 4, 5]$), as shown in Figure 3(b).

We assess the performance of trees constructed using Des-c across various depths and with different cluster sizes (k), comparing the results with the method lacking feature weighting (*no weight*) using the same cluster count, as well as the baseline binary-split method. Figures 3 (a) and (b) showcase the entropy and accuracy, respectively, for visualization purposes, plotting the mean values while omitting the standard deviation as error bars. In Table 2, which presents the primary outcomes reflecting the best performance, we include the standard deviation.

As depicted in Fig.3 (a), entropy consistently improves (i.e., decreases) as the depth increases

Figure 2: Entropy as a function of the number of clusters. Des-c is compared with the same method but without weight (*no weight*). The boxes represent the statistics over the ten folds considered. We include the baseline, which is the entropy calculated by the baseline method. The shaded area corresponds to the standard deviation of the median entropy obtained by the baseline method (the median value is shown with the solid line).

across all methods. A higher cluster count (k) also contributes to improved entropy in our methods, and for high values of k, we even observe



(a) (b)

Figure 3: Performance in the training of the decision trees for the PIMA dataset: (a) shows the entropy at each depth as a function of the tree depth and (b) shows the accuracy (in %) as a function of the tree depth. It is compared Des-c (solid line) to the same method but with no weight (*no weight*) (dotted line) for different numbers of clusters (k), shown in colors. We compared against the baseline (cyan solid line). The values shown correspond to the mean across the folds. We avoid plotting the error bars (standard deviation) for visualization purposes and error values are in line with the values in Table 3.

better entropy at a fixed depth compared to the baseline, suggesting the potential advantages of Des-c. Furthermore, we observe comparable or better performance when incorporating feature weights. This demonstrates that our proposed method effectively reduces entropy as nodes are split and the tree grows, aligning with our expectations. In terms of accuracy (Figure 3 (b)), Des-c consistently achieves higher accuracy than the method without feature weights for a fixed number of clusters (k). All three methods exhibit improved accuracy as the tree depth increases. Notably, for $D \geq 2$, within the overfitting regime, Des-c surpasses the baseline in terms of accuracy. Overall, our findings indicate that, in terms of training performance, the proposed Des-c method competes effectively with the baseline.

7.2.2 Tree inference: performance with test data

In Table 3, we present the results for the three datasets at depths below the over-fitting regime ($D \leq 2$), considering different numbers of clusters when evaluating Des-c. The table displays results corresponding to the minimum k value for which Des-c achieves performance comparable to the baseline. Additionally, we provide a comparison with the baseline, which employs binary splits for all depths. For completeness, we report the accuracy obtained with these specific trees on the training data. Our findings indicate that, across the three datasets, the test accuracy achieved with Des-c closely aligns with the accuracy achieved with the baseline for the depths considered. To approach or match the baseline accuracy, it is necessary to increase the number of clusters beyond binary splits when using Des-c. This explains the higher numbers for tree size, i.e., total number of nodes, utilized by the proposed method.

7.3 Regression

For the regression task, our goal is to minimize variance as the tree nodes are split. The variance of a tree trained at a specific depth D is calculated as follows: $V_D = \sum_{i=1}^{n} f_i Var(\hat{Y}_i)$, where, f_i is the fraction of samples in the i-th node, $\hat{Y}_i$ is the predictor of the sample, Var refers to the variance and n is the total number of leaves in the tree.

7.3.1 Tree construction: performance with training data

Similar to our approach for classification, we start by analyzing the performance at a depth of one. In Figure 4, we compare the variance achieved with Des-c and the same method without weight (labeled as “no weight") as a function of the number of clusters (k), and we compare these results against the baseline. We observe that as we increase the number of clusters, the variance decreases and approaches the baseline. Specifically, for $k = 5$ and $k = 6$, the variance obtained with Des-c overlaps with the baseline. This observation indicates that Des-c becomes increasingly competitive with the baseline as the number of clusters grows.

Dataset	Model	Test Acurracy (%)	Training Accuracy (%)	Tree size	Tree depth	Clusters per depth
PIMA	Baseline	73.51 ± 4.36	73.88 ± 0.51	3	1	2
	Des-c	69.9 ± 6.41	73.39 ± 2.45	3	1	2
	Baseline	74.64 ± 2.81	76.72 ± 1.1	7	2	2
	Des-c	70.34 ± 4.53	77.05 ± 0.81	52.2	2	7
Spambase	Baseline	74.97 ± 12.1	79.44 ± 0.13	3	1	2
	Des-c	75.47 ± 11.78	75.01 ± 11.3	6	1	5
	Baseline	81.89 ± 2.32	82.59 ± 0.13	7	2	2
	Des-c	80.71 ± 9.38	79.61 ± 9.02	27	2	5
Blood	Baseline	77.63 ± 0.82	77.63 ± 0.09	3	1	2
	Des-c	76.87 ± 1.5	77.74 ± 0.18	4	1	3
	Baseline	77.07 ± 2.10	78.05 ± 0.7	7	2	2
	Des-c	77.26 ± 2.05	77.88 ± 0.19	12.8	2	3

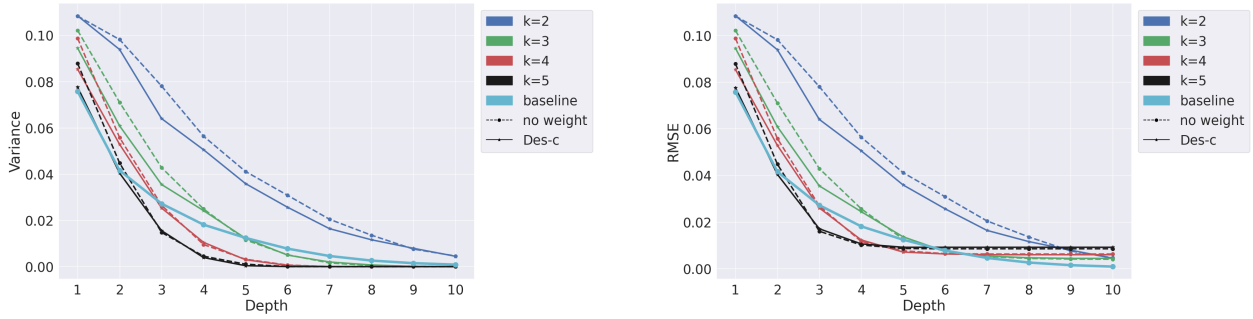
Table 3: Benchmark of binary classification for the PIMA, Spambase and Blood datasets. We compare the proposed method Des-c to the baseline for two tree depths: 1 and 2, below the over-fitting regime. It is shown the mean values and their standard deviations for the test and training accuracy (%). The tree sizes correspond to the average of the sizes among the trees constructed in each fold.

Figure 4: Variance as a function of the number of clusters. It is compared Des-c and the same method but without weight (*no weight*). The boxes correspond to the statistics over the ten folds considered. We include the baseline, which is the variance calculated by the baseline method. The shaded area corresponds to the standard deviation of the variance corresponding to the baseline method (the median value is shown with the solid line).

In Fig. 5, we compare the performance of the decision trees as a function of the tree depth in the training dataset. Similar to the regression scenario, the performance of the models benefits from an increase in depth or the number of clusters. As k increases, we see in Fig. 5 (a) that the variance drops even more rapidly with Des-c as the depth increases than the method with no weight, showing the improvement of tree construction by incorporating the feature weights. Regarding the RMSE, as shown in Figure 5 (b), all three methods experience a reduction in RMSE as the tree depth increases. The RMSE values obtained with Des-c are competitive with those of the baseline. Furthermore, it is evident that the proposed method, Des-c, achieves a greater reduction in RMSE compared to the case without weight. Consequently, in terms of training performance, we can conclude that the proposed method, Des-c, competes effectively with the baseline.

7.3.2 Tree inference: performance with test data

To address a regression task, once we have constructed the tree using the training data, it becomes essential to assign a predictor to each leaf node. This predictor corresponds to the prediction for all the samples allocated to that specific leaf node. The approach involves calculating the average value of the labels associated with the training samples assigned to the leaf node, where $Y_i, i \in n_i$, and n_i represents the training samples assigned to the i-th leaf. Table 4 presents the results for two different tree depths (depths 1 and 2). We assess the trees built with Des-c using varying numbers of clusters and compare them to the baseline, which employs binary splits at all depths. Our report includes the results obtained with "Des-c" using the minimum number of clus-



(a) (b)

Figure 5: Performance in the training of the decision trees for the regression of the Boston housing dataset: (a) shows the variance at given depth as a function of the tree depth and (b) shows the RMSE of the predictors as a function of the tree depth. The performance of Des-c (solid line) is compared to the same method but with no weight (*no weight*) (dotted line) for different numbers of clusters (k), shown in colors. We compared against the baseline (cylon solid line). The values shown correspond to the mean across the folds. The standard deviation is not plotted as error bars to help visualize the trend. In the tables, we report the values with their standard deviation.

ters (k) required to achieve RMSE comparable to the baseline. We also provide the variance observed in the test results, which represents the weighted sum of variances among the test samples assigned to the leaf nodes of the tree.

Our findings indicate that, for the two depths considered, Des-c performs comparably with the baseline, as there is an overlap between RMSE and variance for both methods. However, as previously noted for classification, the proposed method demonstrates improved performance and competitiveness with the baseline as the number of clusters is increased.

8 Conclusion

State-of-the-art decision tree methods rely on axis-parallel splits, using impurity measures for classification and variance reduction for regression. These methods scale polynomially with both the number of training examples N and features d. Existing quantum algorithms achieve quadratic speedups in d but not in N, which is usually more critical for practical speedups. Additionally, they don't address the need for periodic updates to maintain model performance with new data.

We introduce Des-q, a novel quantum algorithm for constructing and retraining decision trees. Once deployed, it can retrain the tree in poly-logarithmic time relative to the number of training examples N. Des-q is designed for binary classification and regression tasks with numerical data.

Des-q leverages the quantum version of the k-means algorithm devised by Kerenidis *et al.* [31], known as q-means, by developing a quantum-supervised clustering algorithm that is utilized to sequentially split the nodes of the tree, from root up to reaching a desired depth. This quantum-supervised clustering algorithm incorporates feature weights in the distance calculation between the training examples and the centroids. To implement feature weighting, we have introduced a quantum algorithm for estimating the Pearson correlation in regression tasks, where the labels are continuous, and the point-biserial correlation in classification tasks, which involve binary labels. This method scales as $\mathcal{O}(1/\epsilon)$ to estimate the feature weights within ϵ precision.

The most significant advantage of our proposed algorithm, Des-q, lies in tree retraining, in particular when dealing with small batches of new data, where $N_{\text{new}} \ll N$. After initially loading the first N examples into the KP-tree data structure with polynomial time complexity in N, adding a new small batch of data only requires time polynomial in N_{new}. This efficient process enables us to build Des-q for tree retraining with poly-logarithmic time complexity in $N + N_{\text{new}} \approx N$.

In terms of the model performance, we have investigated the performance of the classical analog of Des-q on three datasets for binary classification and one for regression. The results demonstrate that our proposed method achieves test performance (accuracy for classification and root-mean-

Method	RMSE	Variance	Tree depth	Tree size	Clusters
Baseline	0.084 ± 0.015	0.081 ± 0.017	1	3	2
Des-c	0.091 ± 0.026	0.084 ± 0.025	1	5	4
Baseline	0.054 ± 0.007	0.049 ± 0.011	2	7	2
Des-c	0.057 ± 0.051	0.037 ± 0.057	2	31	5

Table 4: Performance in test dataset for the regression of the Boston housing dataset. RMSE and the variance are reported. We compare the for the same number of cluster for two tree depths: 1 and 2. They are also compared against the binary-split technique (the baseline) for the same tree sizes and (number of nodes). It is reported the average and the standard deviation across the five folds of both RMSE and variance and the tree size correspond to the average.

square error for regression) that is highly competitive with state-of-the-art axis-parallel split-based methods, which utilize entropy for constructing classification trees and variance reduction for regression trees.

We also remark on the possibility of improving the existing classical algorithms for decision tree construction and retraining by leveraging quantum-inspired classical algorithms, also referred to as *de-quantized* algorithms [58]. This could be relevant in light of the recent dequantization result of the unsupervised clustering algorithm q-means by Doriguello *et al.* [17]. We consider this as a promising direction for future research.

Data and code availability

The results presented in the paper can be reproduced utilizing the data and code available in https://zenodo.org/records/8428525.

Author contribution

R. Yalovetzky, N. Kumar, and C. Li devised the project. N. Kumar and R. Yalovetzky wrote the algorithm for Des-q. N. Kumar, C. Li and R. Yalovetzky did the error analysis for the components of Des-q. R. Yalovetzky devised the plan of carrying out the numerics. R. Yalovetzky, C. Li, and P. Minnsen carried out the numerics and benchmarked Des-q against state-of-art baseline decision trees. M. Pistoia led the overall project. All authors contributed to technical discussions and the writing of the manuscript.

Acknowledgements

The authors thank Shouvanik Chakrabarti, Ruslan Shaydulin, Yue Sun, Dylan Herman, Arthur Rattew and the other colleagues at the Global Technology Applied Research Center of JPMorgan Chase for support and helpful discussions. In addition, the authors thank Kate Stern-Jones and Bryan C Gasche from CIB, JPMorgan Chase for valuable feedback.

References

[1] Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on simplicity in algorithms*, pages 24–32. SIAM, 2020. URL: https://doi.org/10.1137/1.9781611976014.5.

[2] Haldun Akoglu. User's guide to correlation coefficients. *Turkish journal of emergency medicine*, 18(3):91–93, 2018. URL: https://doi.org/10.1016/j.tjem.2018.08.001.

[3] Jonathan Allcock, Jinge Bao, João F Doriguello, Alessandro Luongo, and Miklos Santha. Constant-depth circuits for uniformly controlled gates and boolean functions with application to quantum memory circuits. *arXiv preprint arXiv:2308.08539*, 2023. URL: https://doi.org/10.22331/q-2024-11-20-1530.

[4] David Arthur and Sergei Vassilvitskii. K-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007. URL: https://dl.acm.org/doi/10.5555/1283383.1283494.

[5] Salman Beigi and Leila Taghavi. Quantum speedup based on classical decision trees. *Quantum*, 4:241, March 2020. URL: https://doi.org/10.22331/q-2020-03-02-241.

[6] Fernando Berzal, Juan-Carlos Cubero, Nicolás Marın, and Daniel Sánchez. Building multi-way decision trees with numerical attributes. *Information Sciences*, 165(1-2):73–90, 2004. URL: https://doi.org/10.1016/j.ins.2003.09.018.

[7] Arnon Boneh and Micha Hofri. The coupon-collector problem revisited — a survey of engineering problems and computational methods. *Stochastic Models*, 13(1):39–66, 1997. URL: https://doi.org/10.1080/15326349708807412.

[8] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002. URL: https://doi.org/10.48550/arXiv.quant-ph/0005055.

[9] Gilles Brassard, Peter HØyer, and Alain Tapp. Quantum counting. In *Automata, Languages and Programming*, pages 820–831. Springer Berlin Heidelberg, 1998. URL: https://doi.org/10.1007/bfb0055105.

[10] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Routledge, October 2017. URL: https://doi.org/10.1201/9781315139470.

[11] RS Bucy and RS Diesposti. Decision tree design by simulated annealing. *ESAIM: Mathematical Modelling and Numerical Analysis*, 27(5):515–534, 1993. URL: https://doi.org/10.1051/m2an/1993270505151.

[12] Harry Buhrman, Richard Cleve, John Watrous, and Ronald De Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001. URL: https://doi.org/10.1103/PhysRevLett.87.167902.

[13] Wray Buntine. Learning classification trees. *Statistics and computing*, 2:63–73, 1992. URL: https://doi.org/10.1007/BF01889584.

[14] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: improved regression techniques via faster hamiltonian simulation. *arXiv preprint arXiv:1804.01973*, 2018. URL: https://doi.org/10.4230/LIPIcs.ICALP.2019.33.

[15] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *International Conference on Machine Learning*, pages 1122–1131. PMLR, 2019. URL: https://doi.org/10.48550/arXiv.1902.10660.

[16] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000. URL: https://doi.org/10.1145/347090.347107.

[17] João F Doriguello, Alessandro Luongo, and Ewin Tang. Do you know what q-means? *arXiv preprint arXiv:2308.09701*, 2023. URL: https://doi.org/10.48550/arXiv.2308.09701.

[18] Federico D'Onofrio, Giorgio Grani, Marta Monaci, and Laura Palagi. Margin optimal classification trees. *Computers & Operations Research*, 161:106441, 2024. URL: https://doi.org/10.48550/arXiv.2210.10567.

[19] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical review letters*, 100(16):160501, 2008. URL: https://doi.org/10.1103/PhysRevLett.100.160501.

[20] Tudor Giurgica-Tiron, Iordanis Kerenidis, Farrokh Labib, Anupam Prakash, and William Zeng. Low depth algorithms for quantum amplitude estimation. *Quantum*, 6:745, 2022. URL: https://doi.org/10.22331/q-2022-06-27-745.

[21] Gene V Glass and Julian C Stanley. *Statistical methods in education and psychology*. Prentice-Hall, 1970.

[22] Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation. *npj Quantum Information*, 7(1):52, 2021. URL: https://doi.org/10.1038/s41534-021-00379-1.

[23] Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Information and Compu-*

tation, 126(2):114–122, 1996. URL: https://doi.org/10.1006/inco.1996.0040.

[24] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978. URL: https://doi.org/10.1016/0095-0696(78)90006-2.

[25] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009. URL: https://doi.org/10.1103/PhysRevLett.103.150502.

[26] Raoul Heese, Patricia Bickert, and Astrid Elisa Niederle. Representation of binary classification trees with binary features by quantum circuits. *Quantum*, 6:676, March 2022. URL: https://doi.org/10.22331/q-2022-03-30-676.

[27] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998. URL: https://doi.org/10.1109/34.709601.

[28] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. Spambase. UCI Machine Learning Repository, 1999. URL: https://doi.org/10.24432/C53G6X.

[29] Ragesh Jaiswal. A quantum approximation scheme for k-means. *arXiv preprint arXiv:2308.08167*, 2023. URL: https://doi.org/10.48550/arXiv.2308.08167.

[30] Samuel Jaques and Arthur G Rattew. Qram: A survey and critique. *arXiv preprint arXiv:2305.10310*, 2023.

[31] Iordanis Kerenidis, Jonas Landman, Alessandro Luongo, and Anupam Prakash. q-means: A quantum algorithm for unsupervised machine learning. *Advances in neural information processing systems*, 32, 2019. URL: https://doi.org/10.48550/arXiv.1812.03584.

[32] Iordanis Kerenidis and Alessandro Luongo. Classification of the mnist data set with quantum slow feature analysis. *Physical Review A*, 101(6):062327, 2020. URL: https://doi.org/10.1103/PhysRevA.101.062327.

[33] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675*, 2016. URL: https://doi.org/10.1103/PhysRevLett.100.160501.

[34] Kamil Khadiev, Ilnaz Mannapov, and Liliya Safina. The quantum version of classification decision tree constructing algorithm c5.0, 2019. URL: https://arxiv.org/abs/1907.06840.

[35] Igor Kononenko. Estimating attributes: Analysis and extensions of relief. In *European conference on machine learning*, pages 171–182. Springer, 1994.

[36] Roger J. Lewis. An introduction to classification and regression tree (cart) analysis. 2000. URL: https://api.semanticscholar.org/CorpusID:14981989.

[37] JM Linacre and G Rasch. The expected value of a point-biserial (or similar) correlation. *Rasch Measurement Transactions*, 22(1):1154, 2008.

[38] Zhenyu Liu, Tao Wen, Wei Sun, and Qilong Zhang. A novel multiway splits decision tree for multiple types of data. *Mathematical Problems in Engineering*, 2020:1–12, 2020. URL: https://doi.org/10.1155/2020/7870534.

[39] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013. URL: https://doi.org/10.4230/LIPIcs.ITCS.2017.49.

[40] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

[41] Songfeng Lu and Samuel L. Braunstein. Quantum decision tree classifier. *Quantum Information Processing*, 13(3):757–770, November 2013. URL: https://doi.org/10.1007/s11128-013-0687-5.

[42] Songfeng Lu and Samuel L Braunstein. Quantum decision tree classifier. *Quantum information processing*, 13(3):757–770, 2014. URL: https://doi.org/10.1007/s11128-013-0687-5.

[43] Chaitanya Manapragada, Geoffrey I Webb, and Mahsa Salehi. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1953–1962, 2018. URL: https://doi.org/10.1145/3219819.3220005.

[44] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017. URL: https://doi.org/10.1007/s11128-013-0687-5.

[45] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2(1):1–8, 2016. URL: https://doi.org/10.1038/npjqi.2016.23.

[46] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010. URL: https://doi.org/10.1017/CBO9780511976667.

[47] Andrei Novikov. Pyclustering: Data mining library. *Journal of Open Source Software*, 4(36):1230, apr 2019. doi:10.21105/joss.01230.

[48] Karl Pearson. Vii. note on regression and inheritance in the case of two parents. *proceedings of the royal society of London*, 58(347-352):240–242, 1895. URL: http://www.jstor.org/stable/115794.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL: https://doi.org/10.48550/arXiv.1201.0490.

[50] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986. URL: https://doi.org/10.1007/BF00116251.

[51] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014. URL: https://doi.org/10.1007/BF00993309.

[52] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL: https://doi.org/10.1609/aaai.v29i1.9277.

[53] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. Quantum arithmetic with the quantum fourier transform. *Quantum Information Processing*, 16:1–14, 2017. URL: https://doi.org/10.1007/s11128-017-1603-1.

[54] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, pages 42–47. IEEE, 2013. URL: https://doi.org/10.1109/CTS.2013.6567202.

[55] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021. URL: https://doi.org/10.1109/CTS.2013.6567202.

[56] Himani Sharma, Sunil Kumar, et al. A survey on decision tree algorithms of classification in data mining. *International Journal of Science and Research (IJSR)*, 5(4):2094–2097, 2016.

[57] Matthew Smith, Christian Szongott, Benjamin Henne, and Gabriele Von Voigt. Big data privacy issues in public social media. In *2012 6th IEEE international conference on digital ecosystems and technologies (DEST)*, pages 1–6. IEEE, 2012. URL: https://doi.org/10.1109/DEST.2012.6227909.

[58] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pages 217–228, 2019. URL: https://doi.org/10.1145/3313276.3316310.

[59] Hao Tang, Boning Li, Guoqing Wang, Haowei Xu, Changhao Li, Ariel Barr, Paola Cappellaro, and Ju Li. Communication-efficient quantum algorithm for distributed machine learning. *Phys. Rev. Lett.*, 130:150602, Apr 2023. URL: https://doi.org/10.1103/PhysRevLett.130.150602.

[60] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Com-*

puter Science Department, Trinity College Dublin, 106(2):58, 2004.

[61] Paul E Utgoff. Incremental induction of decision trees. *Machine learning*, 4:161–186, 1989. URL: https://doi.org/10.1023/A:1022699900025.

[62] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv preprint arXiv:1401.2142*, 2014. URL: https://dl.acm.org/doi/abs/10.5555/2871393.2871400.

[63] Min Xu, Pakorn Watanachaturaporn, Pramod K Varshney, and Manoj K Arora. Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment*, 97(3):322–336, 2005. URL: https://doi.org/10.1016/j.rse.2005.05.008.

[64] I-Cheng Yeh. Blood Transfusion Service Center. UCI Machine Learning Repository, 2008. DOI: https://doi.org/10.24432/C5GS39.

[65] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *IJCAI*, pages 3553–3559, 2017. URL: https://doi.org/10.24963/ijcai.2017/497.

[66] Haoran Zhu, Pavankumar Murali, Dzung Phan, Lam Nguyen, and Jayant Kalagnanam. A scalable mip-based method for learning optimal multivariate decision trees. *Advances in neural information processing systems*, 33:1771–1781, 2020. URL: https://doi.org/10.48550/arXiv.2011.03375.

[67] Paul Zikopoulos and Chris Eaton. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

A Ingredients

This section covers the necessary ingredients required to build Des-q.

A.1 Amplitude amplification and estimation

Theorem A.1 (Amplitude amplification [8])**.** *Given the ability to implement a quantum unitary map U and $U^\dagger$ such that $U|0\rangle = \sin\theta\,|x,1\rangle + \cos\theta\,|G,0\rangle$, where $|G\rangle$ is the garbage state, then we can create the state $|x\rangle$ in time $\mathcal{O}(\frac{T(U)}{\sin\theta})$, where $T(U)$ is the time required to implement U and $U^\dagger$.*

Theorem A.2 (Amplitude estimation [8])**.** *Given the ability to implement a quantum unitary map U and $U^\dagger$ such that $U|0\rangle = \sin\theta\,|x,1\rangle + \cos\theta\,|G,0\rangle$, where $|G\rangle$ is the garbage state, then $\sin\theta$ can be estimated up to additive error $\epsilon > 0$ in time $\mathcal{O}(\frac{T(U)}{\epsilon})$, where $T(U)$ is the time required to implement U and $U^\dagger$.*

Suppose we have a unitary U that can implement the mapping

$$U|0\rangle_n|0\rangle = \sqrt{p}\,|x\rangle|1\rangle + \sqrt{1-p}\,|G\rangle|0\rangle\,, \tag{42}$$

where $\sqrt{p} = \sin\theta$. Then for any $\epsilon > 0$, the amplitude estimation algorithm can output $\tilde{\theta}$ such that $|\theta - \tilde{\theta}| \leq \epsilon$ and consequently, the respective probability $\tilde{p} = \sin\tilde{\theta}$ is such that

$$|\tilde{p} - p| \leq 2\pi\sqrt{p(1-p)}\epsilon + (\pi\epsilon)^2 < 2\pi\epsilon + (\pi\epsilon)^2 \sim \mathcal{O}(\epsilon) \tag{43}$$

with probability at least $8/\pi^2$ in $P = \mathcal{O}(1/\epsilon)$ iterations of U and $U^\dagger$. Under the hood, the algorithm relies on the use of quantum phase estimation routine [46] where geometrically increasing powers of the operator $\mathcal{Q} = U\mathcal{S}_0U^\dagger\mathcal{S}_x$ are applied with the operators being $S_0 = \mathbb{I} - 2|0\rangle_{n+1}\langle 0|_{n+1}$ and $\mathcal{S}_x = \mathbb{I} - 2|x,1\rangle\langle x,1|$. This is then followed by the inverse quantum Fourier transform (QFT) to obtain the estimate $\tilde{p}$. We note that subsequent proposals of QFT-free amplitude estimation procedures have also been developed [1, 22, 20].

A.2 Boosting amplitude estimation probability

Amplitude estimation allows us to estimate the value of $\tilde{p}$ within a precision given in Eq 43 with probability $\alpha \geq 8/\pi^2$. We leverage the Lemma 8 from Wiebe *et al.* [62] to boost this success probability to any arbitrary value close to 1.

Theorem A.3 (Majority Evaluation [62])**.** *Let $\mathcal{U}$ be a unitary operation that maps*

$$\mathcal{U} : |0\rangle^{\otimes n} \to \sqrt{a}\,|x,1\rangle + \sqrt{1-a}\,|G,0\rangle \tag{44}$$

for some $1/2 < a \leq 1$ in time T. Then there exists a deterministic quantum algorithm such that for any $\Delta > 0$, the algorithm produces a quantum state $|\Psi\rangle$ which obeys $\| |\Psi\rangle - |0\rangle^{\otimes nk}|x,1\rangle \| \leq \sqrt{2\Delta}$ for some integer k in time

$$\mathcal{O}(T\ln(1/\Delta)). \tag{45}$$

We note that the amplitude estimation final state (pre-measurement) has the form (refer to Eq.42)

$$\sqrt{\alpha}\,|\tilde{p}\rangle|G,1\rangle + \sqrt{1-\alpha}\,|G',0\rangle\,, \tag{46}$$

where $|G\rangle$ and $|G'\rangle$ are garbage states. The aim is to amplify the state $|\tilde{p}\rangle$. We utilize Theorem A.3 by taking $|x,1\rangle = |\tilde{p}\rangle|G,1\rangle$ and $|G,0\rangle = |G',0\rangle$. This is utilized to calculate the distances in Theorem 5.4, as it is also done in the q-means algorithms [31].

Proof.

$$\mathcal{M} : |y_1\rangle \cdots |y_k\rangle |0\rangle \to |y_1\rangle \cdots |y_k\rangle |\tilde{y}\rangle , \tag{47}$$

$$(\sqrt{a}\,|y\rangle + \sqrt{1-a}\left|y^{\perp}\right\rangle)^{\otimes k} := A\,|\Psi\rangle + \sqrt{1-|A|^2}\,|\Phi\rangle , \tag{48}$$

$$\mathcal{M}\left((\sqrt{a}\,|y\rangle + \sqrt{1-a}\left|y^{\perp}\right\rangle)^{\otimes k}|0\rangle^{\otimes n}\right) = A\,|\Psi\rangle\,|y\rangle + \sqrt{1-|A|^2}\,|\Phi\rangle\left|y^{\perp}\right\rangle . \tag{49}$$

$$\begin{aligned}\mathcal{U}^{\dagger\otimes k}\left(A\,|\Psi\rangle\,|y\rangle + \sqrt{1-|A|^2}\,|\Phi\rangle\left|y^{\perp}\right\rangle\right) &= \mathcal{U}^{\dagger\otimes k}\left(A\,|\Psi\rangle\,|y\rangle + \sqrt{1-|A|^2}\,|\Phi\rangle\,|y\rangle\right)\\ &\quad + \mathcal{U}^{\dagger\otimes k}\left(\sqrt{1-|A|^2}(|\Phi\rangle\left|y^{\perp}\right\rangle - |\Phi\rangle\,|y\rangle)\right)\\ &= \left|0^{\otimes nk}\right\rangle|y\rangle + \mathcal{U}^{\dagger\otimes k}\left(\sqrt{1-|A|^2}(|\Phi\rangle\left|y^{\perp}\right\rangle - |\Phi\rangle\,|y\rangle)\right).\end{aligned} \tag{50}$$

$$\|\mathcal{U}^{\dagger\otimes k}\left(A\,|\Psi\rangle\,|y\rangle + \sqrt{1-|A|^2}\,|\Phi\rangle\left|y^{\perp}\right\rangle\right) - \left|0^{\otimes nk}\right\rangle|y\rangle\,\| \leq \sqrt{2(1-|A|^2)}. \tag{51}$$

$$\mathbb{P}(y^{\perp}) = \mathbb{P}(N_y < k/2) \leq \exp\left(-2k(|a| - \frac{1}{2})^2\right) = \Delta. \tag{52}$$

$$k \geq \frac{\ln\left(\frac{1}{\Delta}\right)}{2(|a| - \frac{1}{2})^2}. \tag{53}$$

$$\|\mathcal{U}^{\dagger\otimes k}\left(A\,|\Psi\rangle\,|y\rangle + \sqrt{1-|A|^2}\,|\Phi\rangle\left|y^{\perp}\right\rangle\right) - |0\rangle^{\otimes nk}\,|y\rangle\,\| \leq \sqrt{2\Delta}. \tag{54}$$

$$\mathcal{O}(knT) = \mathcal{O}\left(2Tn\left\lceil\frac{\ln(1/\Delta)}{2(|a| - \frac{1}{2})^2}\right\rceil\right) = \mathcal{O}(T\ln(1/\Delta)), \tag{55}$$

□

A.3 Useful quantum subroutines

Similar to classical boolean subroutines, quantum circuits offer a reversible version of classical subroutines in time linear in the number of qubits. We highlight a few subroutines required in the construction of Des-q.

1. Equality and inequality: For two integers i and j, there is a unitary to check the equality with the mapping

$$|i\rangle\,|j\rangle\,|0\rangle \to |i\rangle\,|j\rangle\,|[i=j]\rangle$$

. If the integers are represented with n bits i.e., $i = i_1 \cdots i_n$ (similarly for j), then one can perform the SWAP-test operation with the ancilla qubit being 0 if the two integers are equal, and 1 otherwise [12]. One can also use the SWAP-test to perform the inequality test for two real numbers x_1 and x_2 by doing the following map

$$|x_1\rangle\,|x_2\rangle\,|0\rangle \to |x_1\rangle\,|x_2\rangle\,|[x_1 \leq x_2]\rangle\,.$$

2. Quantum adder: Another operation of interest is performing the modulo additions of the quantum states, namely

$$|x_1\rangle\,|x_2\rangle \cdots |x_N\rangle\,|0\rangle \to |x_1\rangle\,|x_2\rangle \cdots |x_N\rangle\,|x_1 + x_2 + \cdots x_N \pmod d\rangle\,, \tag{56}$$

where d is the number of qubits used to represent the states $|x_i\rangle$, $i \in [N]$. As highlighted by the authors in [53], this can be done by applying the following operation on the input state

$$IQFT_N \cdot CZ_{1,N} \cdots CZ_{N-2,N} \cdot CZ_{N-1,N} \cdot QFT_N, \tag{57}$$

where $QFT_N, IQFT_N$ implies doing the quantum Fourier transform and inverse quantum Fourier transform respectively on the state $|x_N\rangle$. $CZ_{i,j}$ is the control-phase operation on the states $|x_i\rangle$ and $|x_j\rangle$. This operation takes time $\mathcal{O}(Nd)$ time to perform.

3. Quantum multiplier: One can similarly perform the quantum operation to multiply the contents of N quantum states namely

$$|x_1\rangle |x_2\rangle \cdots |x_N\rangle |0\rangle \to |x_1\rangle |x_2\rangle \cdots |x_N\rangle |x_1 \cdot x_2 \cdot \cdots x_N \pmod d\rangle , \tag{58}$$

where d is the number of qubits used to represent the states $|x_i\rangle$, $i \in [N]$. As highlighted by the authors in [53],this operation can again be done with the application of quantum Fourier transform and controlled weighted sum in time $\mathcal{O}(Nd)$.

4. Non-linear operations One can apply a non-linear function $x \to \phi(x)$ in the quantum domain using the following mapping

$$|x\rangle |0\rangle \to |x\rangle |\phi(x)\rangle .$$

The typical non-linear functions, including $\arcsin x, x^2, \sqrt{x}$, can be applied using Taylor decomposition or other techniques in time linear in the number of qubits of $|x\rangle$.

B Amplitude encoding with oracle QRAM

There are multiple proposals of preparing the state given in Eq 2 or Eq 3 which uses an efficient data loading structure. One such proposal is using direct manipulation of the state generated via quantum random access memory (QRAM) [19]. They are memory models which act as a link between the classical data and quantum states and are capable of answering queries in a quantum superposition. A general QRAM allows having access to the oracle that implements the following unitary

$$\sum_i \alpha_i |i\rangle |0\rangle \to \sum_i \alpha_i |i\rangle |x_i\rangle , \tag{59}$$

whereas the standard classical random access memory provides for the mapping $i \to x_i$.

This oracle-based QRAM model has been extensively studied in quantum complexity literature, for example in algorithms for Grover's search, amplitude amplification, HHL, quantum principal component analysis, and quantum recommendation systems among many others [25, 33, 39]. Physical realizations and architectures of oracle QRAM have been proposed in the literature [19]. For a recent overview of QRAM techniques and implementations, we refer the reader to Jaques *et al.* [30] and Allcock *et al.* [3].

Theorem B.1. *Given a unitary U which takes the state $|0\rangle$ and creates in time $\mathcal{O}(\log d)$ the oracle QRAM encoding quantum state of the vector $x = (x_1, \cdots, x_d)$, then one can perform the following map*

$$\frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |0\rangle \to \frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle \to \frac{1}{\|x\|} \sum_{j=1}^{d} x_j |j\rangle |0\rangle \tag{60}$$

in time $\mathcal{O}(\sqrt{d}\log(d)\frac{\eta^2}{\|x\|})$, where $\eta \geq max(x_j)$.

Proof. Let us apply the conditional rotation on the QRAM state

$$\frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle |0\rangle \to \frac{1}{\sqrt{d}} \sum_{j=1}^{d} |j\rangle |x_j\rangle \left(\frac{x_j}{\eta} |0\rangle + \sqrt{1 - \frac{x_j^2}{\eta^2}} |1\rangle \right) . \tag{61}$$

So if the vector x is normalized then $\eta = 1$. From this, one can simply measure the ancilla bit in the state $|0\rangle$ to end up with te desired state with probability $P(0)$. So this probability is

$$P(0) = \frac{1}{d} \sum_{j=1}^{d} \frac{x_j^2}{\eta^2} = \frac{\|x\|^2}{d\eta^2} . \tag{62}$$

Note that this gives us that one needs $\mathcal{O}(1/P(0))$ samples to have a constant probability of creating the amplitude encoded state. This complexity can be quadratically improved if instead of directly measuring the last qubit, we perform amplitude amplification on it to obtain the amplitude-encoded state. This reduces the complexity of preparing the state from the QRAM state to $\mathcal{O}(1/\sqrt{P(0)})$. Unless in the special cases, the complexity of this method then scales as $\mathcal{O}(\frac{1}{\sqrt{d}})$.

Thus the final state obtained by either directly measuring the last qubit state or by performing amplitude amplification is

$$\frac{1}{\sqrt{d}}\sum_{j=1}^{d} cx_j \left|j\right\rangle \left|x_j\right\rangle \left|0\right\rangle, \tag{63}$$

where c is the proportionality factor. Given that the state must respect the normalization rule, we have that $c = d/\|x\|^2$. This gives us the state

$$\frac{1}{\|x\|}\sum_{j=1}^{d} x_j \left|j\right\rangle \left|x_j\right\rangle. \tag{64}$$

Now in order to get the final amplitude encoding state, one can apply the inverse QRAM unitary state to obtain the desired amplitude encoding state. □

Note that a direct implication of this theorem is the following lemmas

Lemma B.1 (Loading the matrix)**.** *Given a unitary U which takes the state $\left|0\right\rangle$ and creates in time $\mathcal{O}(\log(Nd))$ the oracle QRAM encoding quantum state of the data matrix $X \in \mathbb{R}^{N\times d}$, then one can perform the following map*

$$\frac{1}{\sqrt{Nd}}\sum_{i=1}^{N}\sum_{j=1}^{d}\left|i\right\rangle\left|j\right\rangle\left|0\right\rangle \rightarrow \frac{1}{\sqrt{Nd}}\sum_{i,j}\left|i\right\rangle\left|j\right\rangle\left|x_{ij}\right\rangle \rightarrow \frac{1}{\|X\|_F}\sum_{i=1}^{N}\|x_i\|\left|x_i\right\rangle\left|i\right\rangle \tag{65}$$

in time $\mathcal{O}(\sqrt{Nd}\log(Nd)\frac{\eta^2}{\|X\|_F})$, where $\eta \geq max(x_{ij})$. Where $\left|x_i\right\rangle = \frac{1}{\|x_i\|}\sum_{j=1}^{d} x_{ij}\left|j\right\rangle$ and $\|X\|_F = \sqrt{\sum_{i=1}^{N}\|x_i\|^2}$

Lemma B.2 (Loading the features)**.** *Given a unitary U which takes the state $\left|0\right\rangle$ and creates in time $\mathcal{O}(\log(N))$ the oracle QRAM encoding quantum state of the feature vector $x^{(j)} := (x_{1j}, \cdots, x_{Nj}), j \in [d]$ for the data matrix $X \in \mathbb{R}^{N\times d}$, then one can perform the following map*

$$\frac{1}{\sqrt{N}}\sum_{i=1}^{N}\left|i\right\rangle\left|0\right\rangle \rightarrow \frac{1}{\sqrt{N}}\sum_{i=1}^{N}\left|i\right\rangle\left|x_{ij}\right\rangle \rightarrow \frac{1}{\|x^{(i)}\|}\sum_{i=1}^{N} x_{ij}\left|i\right\rangle \tag{66}$$

in time $\mathcal{O}(\sqrt{N}\log(N)\frac{\eta^2}{\|x^{(i)}\|})$, where $\eta \geq max(x_{ij})$.

An immediate implication of producing amplitude-encoded states via Oracle QRAM routine is that unless in special cases, the time complexity is proportional to square root of the size of the input. Thus, in the aim of having provable exponential speedups in general cases, this method cannot be leveraged. Next, we showcase an alternative memory model with which one can produce the amplitude-encoded states with time complexity logarithm in the size of the input data.

B.1 Estimating Pearson correlation by directly using QRAM-like state

The goal here is to calculate the correlation coefficient defined in Eq 83 using the following QRAM-like states

$$\frac{1}{\sqrt{N}}\sum_{i}^{N}\left|i\right\rangle_n\left|x_i^{(j)}\right\rangle, \quad \frac{1}{\sqrt{N}}\sum_{i}^{N}\left|i\right\rangle_n\left|y_i\right\rangle, \tag{67}$$

where the data $x_i^{(j)}$ and y_i are encoded in the states and $|i\rangle_n$ is an $n \equiv \lceil \log_2(N) \rceil$-qubit (called index qubit hereafter) state $|i_1 i_2 \cdots i_n\rangle$, representing the index of the queried component. This state can be prepared with QRAM, or with a well-designed oracle circuit.

For now, we consider the case where $x_i^{(j)}$ and y_i are binary and can be extended to general floating numbers. To estimate the Pearson correlation coefficient, we are interested in estimating $\sum x_i^{(j)} \cdot y_i$, $\sum x_i^{(j)}$ and $\sum y_i$. Note that $I(x^{(j)}, \mathbb{1}) = x^{(j)} \cdot \mathbb{1} = \sum_i x_i^{(j)}$ and $I(Y, \mathbb{1}) = Y \cdot \mathbb{1} = \sum_i y_i$, where $\mathbb{1}$ is the identity vector. Therefore, the problem is reduced to estimating these inner products. There are several methods for this binary correlation problem. One example is to use the recently-developed quantum bipartite correlator (QBC) algorithm [59] that is based on quantum counting [9].

While the detailed information can be found in [59], here we briefly illustrate the algorithm flow. The idea is to convert the correlation to be estimated into phase information.

As an example, we consider the estimation of $\sum_i^N x_i^{(j)} \cdot y_i$ for evaluation of Pearson correlation coefficient w_j. Firstly, one encodes the information with two qubits q_1 and q_2, that is

$$\frac{1}{\sqrt{N}} \sum_i^N |i\rangle_n |0\rangle_{q_1} |0\rangle_{q_2} \to \frac{1}{\sqrt{N}} \sum_i^N |i\rangle_n \left|x_i^{(j)}\right\rangle_{q_1} |y_i\rangle_{q_2} . \tag{68}$$

Then, a CZ gate between qubits q_1 and q_2 will yield state

$$\frac{1}{\sqrt{N}} \sum_i^N (-1)^{x_i^{(j)} y_i} |i\rangle_n \left|x_i^{(j)}\right\rangle_{q_1} |y_i\rangle_{q_2} . \tag{69}$$

Finally, the qubits q_1 and q_2 are disentangled with index qubits:

$$\frac{1}{\sqrt{N}} \sum_i^N (-1)^{x_i^{(j)} y_i} |i\rangle_n \left|x_i^{(j)}\right\rangle_{q_1} |y_i\rangle_{q_2} \to \frac{1}{\sqrt{N}} \sum_i^N (-1)^{x_i^{(j)} y_i} |i\rangle_n |0\rangle_{q_1} |0\rangle_{q_2} . \tag{70}$$

The above operations, followed by operations $\hat{H}^{\otimes n}(2|0\rangle_n\langle 0|_n - \hat{I})\hat{H}^{\otimes n}$ on index qubits, serve as the Grover operator for quantum counting algorithm. By applying the quantum counting algorithm with the help of register qubits, the correlation $\frac{1}{N}\sum_i^N x_i^{(j)} y_i$ would be extracted.

We remark that the computational complexity for this process would be $O(\frac{logN}{\epsilon})$ with ϵ being the estimation error bound.

C Proof of Lemma 5.1

Lemma C.1. *Let $X \in \mathbb{R}^{N \times d}$ be a given dataset. Then there exists a classical data structure to store the rows of X with the memory and time requirement to create the data structure being $T_{kp} = \mathcal{O}(Nd \log^2(Nd))$ such that, there is a quantum algorithm with access to the data structure which can perform the following unitaries (and also in superposition) in time $T = \mathcal{O}(poly\log(Nd))$*

$$|i\rangle |0\rangle \to |i\rangle \frac{1}{\|x_i\|} \sum_{j=1}^d x_{ij} |j\rangle \tag{71}$$

$$|0\rangle \to \frac{1}{\|X\|_F} \sum_{i=1}^N \|x_i\| |i\rangle \tag{72}$$

Proof. Consider $X \in \mathbb{R}^{N \times d}$ where each row is a d dimensional vector $x_i \in \mathbb{R}^d$, such that $x_i = (x_{i1} \cdots x_{id})$. Then in order to store it in the data structure, we construct N binary-tree data structures $B_i, i \in [N]$ with d leaves. The tree is initially empty and it is updated in an online sequential manner. When a new entry (i, j, x_{ij}) arrives, the leaf node j in the tree B_i is created if it is not present already. The leaf j then stores the value of x_{ij}^2 as well as the sign of x_{ij}. Since there are d leaves in the tree B_i,

the depth of the tree is $\lceil \log d \rceil$. An internal node l of the tree at depth t stores the sum of the values of all the leaves in the subtree rooted at l i.e., the sum of the square amplitudes of the leaves in the subtree. It follows that the root node (depth = 0) of the tree B_i contains the sum of the amplitudes $\sum_{j=1}^{d} x_{ij}^2$. Let us denote the value of the internal node l at depth t of the i-th tree as $B_{i,(t,l)}$, where $l \in \{0,1\}^t$. Then this value can be written as

$$B_{i,(t,l)} = \sum_{j_1 \cdots j_t = l; j_{t+1}, \cdots, j_{\lceil \log d \rceil} \in \{0,1\}} x_{ij}^2 \tag{73}$$

The above equation says that the first t bits of j, written in the binary notation $j \equiv j_1 \cdots j_{\lceil \log d \rceil}$, are fixed to l indicating the depth t. The rest of the bits $(j_{t+1}, \cdots, j_{\lceil \log d \rceil})$ are values in $\{0,1\}$.

A new entry into the tree updates all the nodes of the path from the leaf to the root node, thus updating $\lceil \log d \rceil$ nodes. A caveat with this tree-like structure is that the tree levels are stored as ordered lists and thus the nodes are retrieved in $\mathcal{O}(\log(Nd))$ time in order to be updated. Thus, if one has to update the tree with the entry (i, j, x_{ij}), then the total time required for this is $\mathcal{O}(\log^2(Nd))$ since $\lceil \log d \rceil$ nodes are updated from leaf to the node. Thus the total time to update all the elements of the matrix in the empty tree is $T_{kp} = \mathcal{O}(Nd \log^2(Nd))$. Similarly, the total memory requirement of this tree is $\mathcal{O}(Nd \log^2(Nd))$.

Now we will see how to create the amplitude encoding state given that we have quantum superposition access to this classical data structure. In order to create the amplitude encoding state corresponding to the row $i \in [N]$, we start with the initial state $|0\rangle^{\otimes \lceil \log d \rceil}$, we query the tree B_i and then we perform $\lceil \log d \rceil$ conditional rotations as explained below.

$$\begin{aligned}
\text{Initial state} &= \underbrace{|0\rangle \cdots |0\rangle}_{\lceil \log d \rceil} \\
\text{Conditional rotation (qubit 1)} &= \frac{1}{\sqrt{B_{i,(0,0)}}} (\sqrt{B_{i,(1,0)}} |0\rangle + \sqrt{B_{i,(1,1)}} |1\rangle) \underbrace{|0 \cdots 0\rangle}_{\lceil \log d \rceil - 1} \\
\text{Conditional rotation (qubit 2)} &= \frac{1}{\sqrt{B_{i,(0,0)}}} \Bigg(\sqrt{B_{i,(1,0)}} |0\rangle \frac{1}{\sqrt{B_{i,(1,0)}}} (\sqrt{B_{i,(2,00)}} |0\rangle + \sqrt{B_{i,(2,01)}} |1\rangle) \\
&\quad + \sqrt{B_{i,(1,1)}} |0\rangle \frac{1}{\sqrt{B_{i,(1,1)}}} (\sqrt{B_{i,(2,10)}} |0\rangle + \sqrt{B_{i,(2,11)}} |1\rangle) \Bigg) \underbrace{|0 \cdots 0\rangle}_{\lceil \log d \rceil - 2} \\
&\vdots \\
\text{Conditional rotation (qubit } \lceil \log d \rceil) &= \frac{1}{\sqrt{B_{i,(0,0)}}} \sum_{l=0}^{2^{\lceil \log d \rceil} - 1} \text{sgn}(x_{il}) \sqrt{B_{i,(\lceil \log d \rceil, l)}} |l\rangle \\
&= \frac{1}{\|x_i\|^2} \sum_{j=1}^{d} x_{ij} |j\rangle ,
\end{aligned}$$

where $B_{i,(0,0)} = \sum_{i=1}^{d} x_{ij}^2$ is the value stored in the data structure at the root node. The rotation on the $(t+1)$-th qubit is conditioned on the first t qubits. Also after performing the conditional rotation on the $\lceil \log d \rceil$-th qubit, one also appends the signs of the values stored in the leaf node to prepare the amplitude encoding state. With this method, conditioned on the classical data structure being prepared, the runtime complexity of preparing the amplitude encoded state (for any $i \in [N]$ or also in superposition) is $\mathcal{O}(poly \log(Nd))$.

We also note that we can also create a quantum superposition of the norms of the N examples i.e., implement a unitary $|i\rangle |0\rangle \to |i\rangle |\|X\|\rangle$. This can be done by noting that the roots of all the trees B_i store the values $\|x_i\|^2$. Hence we can construct another binary tree with N leaves such that the leaves

store the amplitudes $\|x_i\|^2, i \in [N]$. Then the root will store the values of $\|X\|_F$. Upon applying the conditional rotations, we can build the final state,

$$|\|X\|\rangle = \frac{1}{\|X\|_F} \sum_{i=1}^{N} \|x_i\| \, |i\rangle \tag{74}$$

□

D Proof of Theorem 5.1

Theorem D.1. *Given access to the amplitude-encoded states for feature vectors $|x^{(j)}\rangle, j \in [d]$ and the label vector $|Y\rangle$ along with their norms $\|x^{(j)}\|$, $\|Y\|$ which are prepared in time $T = \mathcal{O}(poly\log(Nd))$, there exists a quantum algorithm to estimate the Pearson correlation coefficients $\bar{w}_j, \forall j \in [d]$ in time $\mathcal{O}(\frac{Td\eta}{\epsilon^2})$, where $|\bar{w_j} - w_j| \leq \epsilon$, and*

$$\eta = \frac{7 \cdot max\left(\|x^{(j)}\|\|Y\|, \|x^{(j)}\|^2, \|Y\|^2\right)}{N \cdot min\left(\sigma_{x^{(j)}}\sigma_Y, \sigma^2_{x^{(j)}}, \sigma^2_Y\right)},$$

where $\sigma_{x^{(j)}}$ and σ_Y denote the standard deviation for $x^{(j)}$ and Y.

Proof. Using the Lemmas 5.2, 5.3 respectively, we can query in time $T = \mathcal{O}(poly\log(Nd))$ the following states[8]

$$|j\rangle|0\rangle \rightarrow |j\rangle|x^{(j)}\rangle, \;\; |0\rangle \rightarrow |Y\rangle \tag{75}$$

Further, the nature of KP-tree memory model also allows us access to the norms $\|x^{(j)}\|, j \in [d]$, and $\|Y\|$.

We note that the Pearson correlation coefficient w_j can be computed by calculating the inner product between the vectors x^j and Y and also by obtaining the mean of the elements of $x^{(j)}$ and Y. Here we showcase that these operations can be done efficiently if the vectors are encoded in amplitude-encoded states. We start by showcasing the method for computing the inner produce of the two vectors. This can then be easily extended to computing the mean of the vectors.

We start with the initial state

$$|\phi_j\rangle = |j\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle . \tag{76}$$

Then the KP-tree is queried controlled on the second register which results in the mappings $|j\rangle|0\rangle|0\rangle \rightarrow |j\rangle|0\rangle|x^{(j)}\rangle$ and $|j\rangle|1\rangle|0\rangle \rightarrow |j\rangle|1\rangle|Y\rangle$. Thus the state after this controlled rotation operation is given by

$$\frac{1}{\sqrt{2}}|j\rangle\left(|0\rangle|x^{(j)}\rangle + |1\rangle|Y\rangle\right). \tag{77}$$

Applying Hadamard operation on the second register results in the state

$$\frac{1}{2}|j\rangle\left(|0\rangle(|x^{(j)}\rangle + |Y\rangle) + |1\rangle(|x^{(j)}\rangle - |Y\rangle)\right). \tag{78}$$

Now it can be seen that if we measure the second register, the probability of obtaining outcome 1 is

$$p(1) = \frac{1}{2}\left(1 - I(x^{(j)}, Y)\right), \tag{79}$$

where $I(x^{(j)}, Y) = |\langle x^{(j)}|Y\rangle| = 1/(\|x^{(j)}\|\|Y\|)\sum_{i=1}^{N} x_{ij}y_i$. Here we use the fact that $|x^{(j)}\rangle$ and $|Y\rangle$ have only real amplitudes due to the entries of X and Y being real values.

[8]Since the label state $|Y\rangle$ can be queried in time $\mathcal{O}(poly\log(N))$, for simplicity, we consider the time to query both the states $|Y\rangle$ and $\left|x^{(j)}\right\rangle$ to be $T = \mathcal{O}(poly\log(Nd))$

Let us denote the quantity we want to estimate to be $s_j = \sum_{i=1}^{N} x_{ij} y_i$. Using standard Chernoff bounds [44], one can estimate the quantity $\overline{p(1)}$, such that $|\overline{p(1)} - p(1)| \leq \epsilon$ with $\mathcal{O}(1/\epsilon^2)$ copies of the quantum states. This also provides an ϵ-close estimate on the inner product, $|\overline{I(x^{(j)}, Y)} - I(x^{(j)}, Y)| \leq \epsilon$. Further, we can also quantify the error in estimating the quantity s_j with $\mathcal{O}(1/\epsilon^2)$ copies

$$|\overline{s_j} - s_j| \leq \|x^{(j)}\| \|Y\| \epsilon. \tag{80}$$

The next step is to estimate the mean values of the vectors $x^{(j)}$ and Y which are denoted by μ_j and μ_y respectively. Instead of estimating the mean, we estimate the quantities

$$a_j = \sqrt{N}\mu_j = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} x_{ij}, \;\; b = \sqrt{N}\mu_y = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} y_i$$

The quantity a_j can be estimated by computing the inner product between the states $|x^{(j)}\rangle$ and $|u\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle$

$$I(x^{(j)}, u) = \langle x^{(j)} | u \rangle = \frac{1}{\|x^{(j)}\|} \frac{1}{\sqrt{N}} \sum_{i=1}^{N} x_{ij} = \frac{1}{\|x^{(j)}\|} a_j. \tag{81}$$

Similarly, computing the inner product between $|Y\rangle$ and $|u\rangle$ results in estimating the quantity $I(Y, u) = (1/\|Y\|)b$. With $\mathcal{O}(1/\epsilon^2)$ copies of the quantum states, the error in estimating the quantities $I(x^{(j)}, u)$ and $I(Y, u)$ is ϵ. Thus, the error in estimating a_j and b is

$$|\overline{a_j} - a_j| \leq \|x^{(j)}\| \epsilon, \;\; \|\overline{b} - b\| \leq \|Y\| \epsilon \tag{82}$$

Once we have the estimate of s_j, a_j, and b, we can estimate the Pearson correlation coefficient, since it can be expressed as

$$\begin{aligned} w_j &= \frac{\sum_{i=1}^{N} (x_{ij} - \mu_j)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{N} (x_{ij} - \mu_j)^2} \sqrt{\sum_{i=1}^{N} (y_i - \mu_y)^2}} \\ &= \frac{s_j - a_j b \left(2 - \frac{1}{N}\right)}{\sqrt{\|x^{(j)}\|^2 - a_j^2} \sqrt{\|Y\|^2 - b^2}} \equiv \frac{\text{Num}}{\text{Den}} \end{aligned} \tag{83}$$

Using the error estimates in equations Eq 80 and Eq 82, we can quantify the error in the estimation of the Pearson correlation coefficient. Prior to that, let us estimate the error in the numerator

$$\begin{aligned} \delta\text{Num} &= \left|\frac{\partial \text{Num}}{\partial s_j}\right| |\overline{s_j} - s_j| + \left|\frac{\partial \text{Num}}{\partial a_j}\right| |\overline{a_j} - a_j| + \left|\frac{\partial \text{Num}}{\partial b}\right| |\overline{b} - b| \\ &\leq (\|x^{(j)}\| \|Y\| + \left(2 - \frac{1}{N}\right) b \|x^{(j)}\| + \left(2 - \frac{1}{N}\right) a_j \|Y\|) \epsilon \\ &\leq 5 \|x^{(j)}\| \|Y\| \epsilon, \end{aligned} \tag{84}$$

where to go from the second to final step, we use the Cauchy-Swartz inequality to bound $b \leq \|Y\|$ and $a_j \leq \|x^{(j)}\|$. Similarly, the error in the denominator is given by

$$\begin{aligned} \delta\text{Den} &= \frac{\sqrt{N}\mu_j \sqrt{\|Y\|_2^2 - b^2}}{\sqrt{\|x^{(j)}\|^2 - a_j^2}} |\overline{a_j} - a_j| + \frac{\sqrt{N}\mu_y \sqrt{\|x^{(j)}\|_2^2 - a_j^2}}{\sqrt{\|Y\|^2 - b^2}} |\overline{b} - b| \\ &\leq \left(\frac{\sqrt{N}\mu_j \sqrt{\|Y\|^2 - b^2}}{\sqrt{\|x^{(j)}\|^2 - a_j^2}} \|x^{(j)}\| + \frac{\sqrt{N}\mu_y \sqrt{\|x^{(j)}\|^2 - a_j^2}}{\sqrt{\|Y\|^2 - b^2}} \|Y\| \right) \epsilon. \end{aligned} \tag{85}$$

Using the uncertainty in both the numerator and the denominator, we can upper-bound the error in the correlation coefficient estimation

$$
\begin{aligned}
\delta w_j &= |\overline{w_j} - w_j| = \delta\text{Num}/\text{Den} + \delta\text{Den}\cdot\text{Num}/\text{Den}^2 \\
&\leq \left(5\|x^{(j)}\|\|Y\|/\text{Den} + \left(\frac{\sqrt{N}\mu_j\sqrt{\|Y\|^2-\mu_y^2N}}{\sqrt{\|x^{(j)}\|^2-\mu_j^2N}}\|x^{(j)}\| + \frac{\sqrt{N}\mu_y\sqrt{\|x^{(j)}\|^2-\mu_j^2N}}{\sqrt{\|Y\|^2-\mu_y^2N}}\|Y\|\right)\cdot w_j/\text{Den}\right)\epsilon \\
&= \left(\frac{5\|x^{(j)}\|\|Y\|}{N\sigma_{x^{(j)}}\sigma_Y} + \left(\frac{\sqrt{N}\mu_j\sigma_Y\|x^{(j)}\|}{\sigma_{x^{(j)}}} + \frac{\sqrt{N}\mu_y\sigma_{x^{(j)}}\|Y\|}{\sigma_Y}\right)\cdot\frac{w_j}{N\sigma_{x^{(j)}}\sigma_Y}\right)\epsilon \\
&= \left(\frac{5\|x^{(j)}\|\|Y\|}{N\sigma_{x^{(j)}}\sigma_Y} + \frac{w_j\mu_j\|x^{(j)}\|}{\sqrt{N}\sigma^2_{x^{(j)}}} + \frac{w_j\mu_y\|Y\|}{\sqrt{N}\sigma_Y^2}\right)\epsilon \\
&= \left(\frac{5\|x^{(j)}\|\|Y\|}{N\sigma_{x^{(j)}}\sigma_Y} + \frac{w_ja_j\|x^{(j)}\|}{N\sigma^2_{x^{(j)}}} + \frac{w_jb\|Y\|}{N\sigma_Y^2}\right)\epsilon \\
&\leq \frac{7\cdot\max\left(\|x^{(j)}\|\|Y\|, \|x^{(j)}\|^2, \|Y\|^2\right)}{N\cdot\min\left(\sigma_{x^{(j)}}\sigma_Y, \sigma^2_{x^{(j)}}, \sigma_Y^2\right)}\epsilon
\end{aligned}
\tag{86}
$$

where $\sigma_{x^{(j)}}$ and σ_Y denote the standard deviation for $x^{(j)}$ and Y, respectively and in the last line use the we use the Cauchy-Swartz inequality to bound $b \leq \|Y\|$ and $a_j \leq \|x^{(j)}\|$, followed by the fact that $w_j \leq 1$.

Now, in order to bound $\delta w_j \leq \epsilon$, for any desired ϵ, the total number of copies the quantum state required is $\mathcal{O}(\eta^2/\epsilon^2)$, where η is

$$
\eta = \frac{7\cdot\max\left(\|x^{(j)}\|\|Y\|, \|x^{(j)}\|^2, \|Y\|^2\right)}{N\cdot\min\left(\sigma_{x^{(j)}}\sigma_Y, \sigma^2_{x^{(j)}}, \sigma_Y^2\right)}. \tag{87}
$$

Given that the preparation of each quantum state takes time $T = \mathcal{O}(poly\log(Nd))$, the total time complexity of estimating the correlation coefficient individually for every feature vector $j \in [d]$ is $\mathcal{O}\left(\frac{Td\eta}{\epsilon^2}\right)$. This completes the proof. □

E Proof of Theorem 5.2

Theorem E.1. *Given access to the amplitude-encoded states for feature vectors $|x^{(j)}\rangle, j \in [d]$ and the label vector $|Y\rangle$ along with their norms $\|x^{(j)}\|$, $\|Y\|$ which are prepared in time $T = \mathcal{O}(poly\log(Nd))$, there exists a quantum algorithm to estimate each Pearson correlation coefficient $\bar{w}_j, j \in [d]$ in time $T_w = \mathcal{O}(\frac{Td\eta}{\epsilon})$, where $\|\bar{w}_j - w_j| \leq \epsilon$, and*

$$
\eta = \frac{7\cdot max\left(\|x^{(j)}\|\|Y\|, \|x^{(j)}\|^2, \|Y\|^2\right)}{N\cdot min\left(\sigma_{x^{(j)}}\sigma_Y, \sigma^2_{x^{(j)}}, \sigma_Y^2\right)},
$$

where $\sigma_{x^{(j)}}$ and σ_Y denote the standard deviation for $x^{(j)}$ and Y.

Proof. In Theorem D.1, one could estimate the inner product by measuring the third register in Eq 78 which gives the outcome 1 with probability $p(1)$ as denoted in Eq 79. Indeed as we show below, instead of measuring the third register, if one applies amplitude estimation on the state in Eq 78 (as highlighted in Appendix A.1), we can reduce the time complexity of estimating the Pearson correlation coefficient quadratically in the ϵ, where $|\overline{w_j} - w_j| \leq \epsilon, j \in [d]$.

The idea is that the state $|1\rangle\left(|x^{(j)}\rangle - |Y\rangle\right)$ can be rewritten as $|z_{jY}, 1\rangle$ (by swapping the registers), and hence Eq 78 has the following mapping

$$
|j\rangle|0\rangle|0\rangle \to |j\rangle\left(\sqrt{p(1)}|z_{jY}, 1\rangle + \sqrt{1-p(1)}\,|G, 0\rangle\right), \tag{88}
$$

where G is some garbage state.

Now it is clear that this is the form of the input for amplitude estimation [8] algorithm where the task is to estimate the unknown coefficient $\sqrt{p(1)}$. Applying amplitude estimation results in the output state (prior to measurement)

$$U : |j\rangle |0\rangle \rightarrow |j\rangle \left(\sqrt{\alpha}|\overline{p(1)}, G', 1\rangle + \sqrt{1-\alpha}|G^{''}, 0\rangle\right), \tag{89}$$

where $G', G^{''}$ are garbage registers. The above procedure requires $\mathcal{O}(1/\epsilon)$ iterations of the unitary U (and its transpose) to produce the state such that $|\overline{p(1)} - p(1)| \leq \epsilon$. Measuring the above state results in the estimation of $\overline{p(1)}$ with a constant probability $\alpha \geq 8/\pi^2$. From this, it becomes clear that the quantity $s_j = \sum_{i=1}^{N} x_{ij} y_i$ can be estimated to an accuracy Eq 80 in $\mathcal{O}(1/\epsilon)$ iterations of U.

Similar procedure applies to estimating the quantities $\overline{a_j}$ and $\overline{b}$ to accuracy as in Eq 82 in $\mathcal{O}(1/\epsilon)$ iterations of U.

Thus following the error analysis procedure in Method-1, the Pearson correlation coefficient can be estimated to ϵ precision with time complexity

$$T_w = \mathcal{O}\left(\frac{Td\eta}{\epsilon}\right), \tag{90}$$

where η is defined in Eq 87. □

F Proof of Theorem 5.4

Theorem F.1. *Given quantum access to the dataset X in time $T = \mathcal{O}(poly\log(Nd))$ and quantum access to the weighted centroids $c_{1,w}, \cdots, c_{k,w}$ in time $\mathcal{O}(poly\log(kd))$, then for any $\Delta > 0$ and $\epsilon_1 > 0$, there exists a quantum algorithm such that*

$$\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle \otimes_{j\in[k]} (|j\rangle |0\rangle) \rightarrow \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle \otimes_{j\in[k]} (|j\rangle \left|\overline{I_w(x_i, c_j)}\right\rangle), \tag{91}$$

where $I_w(.)$ is the weighted inner product between the two vectors and $|\overline{I_w(x_i, c_j)} - I_w(x_i, c_j)| \leq \epsilon_1$ with probability at least $1 - 2\Delta$, in time $T_{cd} = \mathcal{O}(Tk\frac{\eta_1}{\epsilon_1}\log(1/\Delta))$, where $\eta_1 = max_i(\|x_i\|^2)$.

Proof. We prove the theorem for the case of estimating the weighted inner product between the vector x_i and c_j within ϵ_1 precision with a probability at least $1 - 2\Delta$ i.e.,

$$|i\rangle |j\rangle |0\rangle \rightarrow |i\rangle |j\rangle \left|\overline{I_w(x_i, c_j)}\right\rangle. \tag{92}$$

The general result in Eq 15 then follows in a straightforward manner.

Similarly to the Pearson correlation estimation in Method-1 of Sec 5.2.2, we start with the initial state

$$|\phi_j\rangle = |i\rangle |j\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |0\rangle. \tag{93}$$

Then the KP-tree is queried controlled on the third register which results in the mappings $|i\rangle |j\rangle |0\rangle |0\rangle \rightarrow |i\rangle |j\rangle |0\rangle |x_i\rangle$ and $|i\rangle |j\rangle |1\rangle |0\rangle \rightarrow |i\rangle |j\rangle |1\rangle |c_{j,w}\rangle$. Thus the state after this controlled rotation operation is given by

$$|i\rangle |j\rangle \frac{1}{\sqrt{2}} (|0\rangle |x_i\rangle + |1\rangle |c_{j,w}\rangle). \tag{94}$$

Applying Hadamard operation on the third register results in the state

$$|i\rangle |j\rangle \left(\frac{1}{2} |0\rangle (|x_i\rangle + |c_{j,w}\rangle) + \frac{1}{2} |1\rangle (|x_i\rangle - |c_{j,w}\rangle)\right). \tag{95}$$

From the above equation, it can be seen that the probability of obtaining the third register's measurement outcome to be 1 is

$$p(1) = \frac{1}{2}(1 - \langle x_i | \, c_{j,w}\rangle) = \frac{1}{2}(1 - I(|x_i\rangle, |c_{j,w}\rangle)), \tag{96}$$

where $I(|x_i\rangle, |c_{j,w}\rangle)$ is the normalized inner product between the two states $|x_i\rangle$ and $|c_{j,w}\rangle$. Note that $I_w(x_i, c_j) = \|x_i\|\|c_{j,w}\| I(|x_i\rangle, |c_{j,w}\rangle)$.

Now, as we had shown in Theorem E.1, instead of directly measuring the third register, we can apply amplitude estimation procedure as method in Appendix A.1 followed by majority evaluation procedure in Appendix A.2 to allow us to estimate the distance $|\overline{I(|x_i\rangle, |c_{j,w}\rangle)} - I(|x_i\rangle, |c_{j,w}\rangle)| \leq \epsilon_1$ with probability at least $1 - 2\Delta$.

The core idea is that after swapping the third and fourth registers of the state in Eq 95, it can be rewritten as

$$U : |i\rangle |j\rangle |0\rangle |0\rangle \rightarrow |i\rangle |j\rangle \left(\sqrt{p(1)} \, |z_{ij}, 1\rangle + \sqrt{1 - p(1)} \, |G, 0\rangle\right), \tag{97}$$

where G is some garbage state. We note that the above operation takes time $\mathcal{O}(poly\log(kd))$ (for preparation of states $|c_{j,w}\rangle$) and $T = \mathcal{O}(poly\log(Nd))$ for preparing the states $|x_i\rangle$. Thus the total time taken to perform the operation in Eq 97 is

$$T_U = \mathcal{O}(poly\log(kd)) + T \approx T$$

given that N is much larger than k.

Next, after applying the amplitude estimation (prior to measurement), we obtain the state

$$|i\rangle |j\rangle \left(\sqrt{\alpha}|\arcsin\sqrt{\overline{p(1)}}\rangle \, |G', 1\rangle + \sqrt{1 - \alpha} \, |G'', 0\rangle\right), \tag{98}$$

where G', G'' are garbage registers and $\overline{p(1)}$ is the ϵ_1 approximation to $p(1)$.

Subsequently, on the third register, we apply the quantum non-linear operation subroutines $|a\rangle |0\rangle \rightarrow |a\rangle |\sin(a)\rangle$ followed by $|\sin(a)\rangle |0\rangle \rightarrow |\sin(a)\rangle |\sin^2(a)\rangle$ as mentioned in Sec A.3. This is followed by applying the quantum adder subroutine, again mentioned in Sec A.3, to apply $|\sin^2(a)\rangle |1\rangle \rightarrow |\sin^2(a)\rangle |1 - 2\sin^2(a)\rangle$. Here $a = \arcsin\overline{p(1)}$. This results in the state

$$|i\rangle |j\rangle \left(\sqrt{\alpha}|1 - 2\overline{p(1)}\rangle |G', 1\rangle + \sqrt{1 - \alpha}|G'', 0\rangle\right) := |i\rangle |j\rangle \left(\sqrt{\alpha}|\overline{I(|x_i\rangle, |c_{j,w}\rangle)}\rangle |G', 1\rangle + \sqrt{1 - \alpha} \, |G'', 0\rangle\right). \tag{99}$$

Subsequently, we multiply the contents of the third register by $\|x\|\|c_{j,w}\|$ to obtain the state (note that this can be done using the quantum multiplier subroutine mentioned in Sec A.3)

$$|i\rangle |j\rangle \left(\sqrt{\alpha}|\overline{I_w(x_i, c_j)}\rangle |G', 1\rangle + \sqrt{1 - \alpha}|G'', 0\rangle\right). \tag{100}$$

The above procedure of amplitude estimation followed by the quantum subroutines takes time $\mathcal{O}(T\frac{\|x_w\|\|c_j\|}{\epsilon_1})$, where the extra factor $\|x_i\|\|c_{j,w}\|$ comes in because the amplitude estimation allows for the estimation of unnormalized values $|\overline{I(|x_i\rangle, |c_{j,w}\rangle)} - I(|x_i\rangle, |c_{j,w}\rangle)| \leq \epsilon_1$. In order to get the estimation of the normalized values and still keep the error bound to ϵ_1, the time complexity gets multiplied by the factor $\|x_i\|\|c_{j,w}\|$.

We can now boost the success probability of obtaining the inner product state from the value α to any desired value $1 - 2\Delta$ by applying the majority evaluation procedure in Sec A.2 using $L = \mathcal{O}(\ln(1/\Delta))$ copies of state in Eq 100. This results in the state

$$\| \, |\Psi\rangle_{ij} - |0\rangle^{\otimes L} \, |\overline{I_w(x_i, c_j)}\rangle |G', 1\rangle \| \leq \sqrt{2\Delta}. \tag{101}$$

The run time of this procedure is $\mathcal{O}(\frac{T\|x_i\|\|c_{j,w}\|}{\epsilon_1} \ln(1/\Delta))$.

With this, the proof of the theorem follows straightforwardly by noting that the norm of the weighted centroids $\|c_{j,w}\| \leq \|w\|\|c_j\|$. Given that $\|w\| = 1$, this implies that the norm of the weighted centroids would be less than the maximum norm of the k original centroids. Further, the norm of the centroid would always be less than the maximum norm of all the weighted examples in the dataset. This implies that $\|x_i\|\|c_{j,w}\| \leq \max_i(\|x_i\|^2) = \eta_1$. □

G Proof of Theorem 5.6

Theorem G.1. *Given access to the characteristic vector states $|\xi_j\rangle$ $\forall j \in [k]$ where each state is prepared in time $T_\xi = \mathcal{O}(T_l \log k)$ and the amplitude-encoded states for feature vectors $|x^{(l)}\rangle$ $\forall l \in [d]$ which can be prepared in time $T = \mathcal{O}(poly\log(Nd))$, there exists a quantum algorithm to obtain the updated centroid vectors $\overline{c_1}, \cdots, \overline{c_k}$ such that $\|\overline{c_j} - c_j\|_\infty \leq \epsilon_2$ $\forall j \in [k]$ in time*

$$T_{sup\text{-}cluster} = \mathcal{O}\left(\frac{T_\xi k^{\frac{3}{2}}}{d}\epsilon_2\right),$$

where $c_j = X^T\xi_j$ is the true mean of the weighted examples in the cluster and thus the true updated centroid vector of c_j and $\eta_2 = max_{l\in[d]}\|x^{(l)}\|$.

Proof. We first highlight the method and time complexity for estimating the centroid vector c_j for a given $j \in [k]$. The total time complexity for the rest of $k-1$ centroids is just repeating the above procedure k times for different characteristic vector states.

Using Lemma 5.2, we can query the following data feature state in time $T = \mathcal{O}(poly\log(Nd))$

$$|l\rangle|0\rangle \to |l\rangle|x^{(l)}\rangle. \tag{102}$$

Also in time $T_\xi = \mathcal{O}(T_l \log k)$, we obtain the state $|\xi_j\rangle$. Since the centroid vector c_j is d-dimensional, our approach is to estimate each component c_{jl}, $l \in [d]$ separately within ϵ_2 precision. This would allow us to estimate the centroid vector such that $\|\overline{c_j} - c_j\|_\infty \leq \epsilon_2$.

Before describing our method, it can be seen that the l-th component of the centroid is $c_{jl} = \frac{1}{|\mathcal{C}_j|}\sum_{i\in\mathcal{C}_i} x_{il}$. Now, this value can be computed from the inner product of the quantum states $|x^{(l)}\rangle$ and $|\xi_j\rangle$ as

$$I(|x^{(l)}\rangle, |\xi_j\rangle) = \langle x^{(l)}|\xi_j\rangle = \frac{1}{\sqrt{|\mathcal{C}_j|}\|x^{(l)}\|}\sum_{i\in\mathcal{C}_j} x_{il} = \frac{\sqrt{|\mathcal{C}_j|}}{\|x^{(l)}\|}c_{jl}. \tag{103}$$

Thus, the estimation of the quantity c_{jl} amounts to estimating the inner product of the two quantum states mentioned above. We now start with the initial state

$$|\phi_l\rangle = |l\rangle|j\rangle\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle \tag{104}$$

Then the KP-tree is queried controlled on the third register which results in the mappings $|l\rangle|j\rangle|0\rangle|0\rangle \to |l\rangle|j\rangle|0\rangle|x^{(l)}\rangle$ and $|l\rangle|j\rangle|1\rangle|0\rangle \to |l\rangle|j\rangle|1\rangle|\xi_j\rangle$. Thus the state after this controlled rotation operation is given by

$$\frac{1}{\sqrt{2}}|l\rangle|j\rangle\left(|0\rangle|x^{(l)}\rangle + |1\rangle|\xi_j\rangle\right) \tag{105}$$

Applying Hadamard operation on the third qubit results in the state

$$\frac{1}{2}|l\rangle|j\rangle\left(|0\rangle(|x^{(l)}\rangle + |\xi_j\rangle) + |1\rangle(|x^{(l)}\rangle - |\xi_j\rangle)\right) \tag{106}$$

Now, the state $|1\rangle(|x^{(l)}\rangle - |\xi_j\rangle)$ can be rewritten as $|z_{lj}, 1\rangle$ (by swapping the registers), and hence Eq 106 has the following mapping

$$|l\rangle|j\rangle|0\rangle|0\rangle \to |l\rangle|j\rangle\left(\sqrt{p(1)}|z_{lj}, 1\rangle + \sqrt{1-p(1)}|G, 0\rangle\right), \tag{107}$$

where G is some garbage state and $p(1)$ is the probability of obtaining outcome 1 when the third register of the state in Eq 106 is measured i.e.,

$$p(1) = \frac{1}{2}\left(1 - I(|x^{(l)}\rangle, |\xi_j\rangle)\right). \tag{108}$$

Now it is clear that this is the form of the input for amplitude estimation [8] algorithm where the task is to estimate the unknown coefficient $\sqrt{p(1)}$. Applying amplitude estimation results in the output state (before measurement)

$$U : |l\rangle |j\rangle |0\rangle \to |l\rangle |j\rangle \left(\sqrt{\alpha}|\overline{p(1)}, G', 1\rangle + \sqrt{1-\alpha}|G^{''}, 0\rangle\right), \tag{109}$$

where $G', G^{''}$ are garbage registers. The above procedure requires $\mathcal{O}(1/\epsilon_2)$ iterations of the unitary U (and its transpose) to produce the state such that $|\overline{p(1)} - p(1)| \leq \epsilon_2$. Measuring the above state results in the estimation of $\overline{p(1)}$ with a constant probability $\alpha \geq 8/\pi^2$.

From this, it becomes clear that we can also get an ϵ_2 estimate on the inner product with $\mathcal{O}(1/\epsilon_2)$ iterations of U. This results in the estimation of c_{jl} with accuracy

$$|\overline{c_{jl}} - c_{jl}| \leq \frac{\|x^{(l)}\|}{\sqrt{|\mathcal{C}_j|}}\epsilon_2 = \epsilon_2'. \tag{110}$$

Denoting ϵ_2' as ϵ_2, the total time required to estimate the value c_{jl} is the time to load the states $|x^{(l)}\rangle$ and $|\xi_j\rangle$ and the subsequent time to perform the inner product estimation between them

$$\begin{aligned} T_{c_{jl}} &= \mathcal{O}\left(\frac{(T_\xi + T)\|x^{(l)}\|}{\sqrt{|\mathcal{C}_j|}\epsilon_2}\right) \\ &= \mathcal{O}\left(\frac{T_\xi\|x^{(l)}\|}{\sqrt{|\mathcal{C}_j|}\epsilon_2}\right) \\ &= \mathcal{O}\left(\frac{T_\xi\sqrt{N}}{\sqrt{\frac{N}{k}}\epsilon_2}\right) \\ &= \mathcal{O}\left(\frac{T_\xi\sqrt{k}}{\epsilon_2}\right), \end{aligned} \tag{111}$$

where we use the assumption that X has bounded constants entries and therefore $\|x^{(l)}\| = \mathcal{O}(\|x^{(l)}\|_\infty\sqrt{N}) = \mathcal{O}(\sqrt{N})$ and the assumption of well-clusterable datasets introduced by Kerenidis *et al.* [31] that each cluster has at least $\Omega(\frac{N}{k})$ samples, i.e., $|\mathcal{C}_j| = \Omega(\frac{N}{k})$. Note this assumption is crucial to remove the dependency with $\sqrt{N}$ that comes from the linear scaling with $\|x^{(l)}\|$.

Now to calculate the other elements $c_{jl'}$ of the vector c_j, we repeat the inner product estimation procedure between $\left|x^{(l')}\right\rangle$ and $|\xi_j\rangle$ which requires the time complexity $\mathcal{O}\left(\frac{T_\xi\sqrt{k}}{\epsilon_2}\right)$. Thus the total time taken to estimate the vector c_j can be bounded by $T_{c_j} = \mathcal{O}\left(\frac{T_\xi d\sqrt{k}}{\epsilon_2}\right)$.

We repeat the above procedure k times to get all the updated centroid vectors $c_1, \cdots, c_k$ within infinity norm ϵ_2 precision with the total time complexity

$$T_{\text{sup-cluster}} = \sum_{j\in[k]} T_{c_j} = \mathcal{O}\left(\frac{T_\xi k^{\frac{3}{2}}}{d}\epsilon_2\right), \tag{112}$$

where sup-cluster stands for supervised clustering. This completes the proof. □

H Proof of Theorem 5.7

Theorem H.1. *Given access to the characteristic vector states $|\xi_j\rangle$ $\forall j \in [k^D]$ where each state is prepared in time $T_\xi = \mathcal{O}(T_l^{k^D} D \log k)$ and the amplitude-encoded states label superposition state $|Y\rangle$*

which is be prepared in time $\mathcal{O}(poly\log(N))$*, there exists a quantum algorithm to obtain the mean label values* $\{\overline{label_1}, \cdots, \overline{label_{k^D}}\}$ *such that* $|\overline{label_j} - label_j| \leq \epsilon_3$*,* $\forall j \in [k^D]$ *in time*

$$T_{\textit{leaf-label}} = \mathcal{O}\left(\frac{T_\xi k^{\frac{5D}{2}}}{\epsilon_3}\right),$$

where $label_j$ *is the true label mean of the examples in the cluster as given in Eq 27.*

Proof. We start with the initial state

$$|\phi_l\rangle = |j\rangle \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle. \tag{113}$$

We query the states $|\xi_j\rangle$ and $|Y\rangle$ with the index $|j\rangle$ controlled on the second register which results in the mappings $|j\rangle|0\rangle|0\rangle \to |j\rangle|0\rangle|Y\rangle$ and $|j\rangle|1\rangle|0\rangle \to |j\rangle|1\rangle|\xi_j\rangle$. Thus the state after this controlled rotation operation is given by

$$\frac{1}{\sqrt{2}}|j\rangle(|0\rangle|Y\rangle + |1\rangle|\xi_j\rangle). \tag{114}$$

Applying Hadamard operation on the third qubit results in the state

$$\frac{1}{2}|j\rangle(|0\rangle(|Y\rangle + |\xi_j\rangle) + |1\rangle(|Y\rangle - |\xi_j\rangle)). \tag{115}$$

Now, the state $|1\rangle(|Y\rangle - |\xi_j\rangle)$ can be rewritten as $|z_{lj}, 1\rangle$ (by swapping the registers), and hence Eq 115 has the following mapping

$$|j\rangle|0\rangle|0\rangle \to |j\rangle\left(\sqrt{p(1)}|z_{lj}, 1\rangle + \sqrt{1-p(1)}|G, 0\rangle\right), \tag{116}$$

where G is some garbage state and $p(1)$ is the probability of obtaining outcome 1 when the third register of the state in Eq 115 is measured i.e.,

$$p(1) = \frac{1}{2}(1 - I(|Y\rangle, |\xi_j\rangle)) \tag{117}$$

Now it is clear that this is the form of the input for amplitude estimation [8] algorithm where the task is to estimate the unknown coefficient $\sqrt{p(1)}$. Applying amplitude estimation results in the output state (before measurement)

$$U : |j\rangle|0\rangle \to |j\rangle\left(\sqrt{\alpha}|\overline{p(1)}, G', 1\rangle + \sqrt{1-\alpha}|G'', 0\rangle\right), \tag{118}$$

where G', G'' are garbage registers. The above procedure requires $\mathcal{O}(1/\epsilon_3)$ iterations of the unitary U (and its transpose) to produce the state such that $|\overline{p(1)} - p(1)| \leq \epsilon_3$. Measuring the above state results in the estimation of $\overline{p(1)}$ with a constant probability $\alpha \geq 8/\pi^2$.

From this, it becomes clear that we can also get an ϵ_3 estimate on the inner product with $\mathcal{O}(1/\epsilon_3)$ iterations of U. Using Eq 28, this results in the estimation of $label_j$ with accuracy

$$|\overline{label_j} - label_j| \leq \frac{\|Y\|}{\sqrt{|\mathcal{C}_{j,D}|}}\epsilon_3 = \epsilon_3'. \tag{119}$$

Denoting ϵ_3' as ϵ_3, the total time required to estimate the value $label_j$ is the time to load the states

$|Y\rangle$ and $|\xi_j\rangle$ and the subsequent time to perform the inner product estimation between them

$$\begin{aligned} T_{\text{label}_j} &= \mathcal{O}\left(\frac{(T_\xi + \mathcal{O}(poly\log(N)))\|Y\|}{\sqrt{|\mathcal{C}_{j,D}|}\epsilon_3}\right) \\ &= \mathcal{O}\left(\frac{T_\xi\|Y\|}{\sqrt{|\mathcal{C}_{j,D}|}\epsilon_3}\right) \\ &= \mathcal{O}\left(\frac{T_\xi\sqrt{N}}{\sqrt{\frac{N}{k^D}}}\epsilon_3\right) \\ &= \mathcal{O}\left(\frac{T_\xi k^{\frac{D}{2}}}{\epsilon_3}\right), \end{aligned} \tag{120}$$

where we use the assumption that Y has bounded constants entries and therefore $\|Y\| = \mathcal{O}(\|Y\|_\infty\sqrt{N}) = \mathcal{O}(\sqrt{N})$. We also use the assumption of well-clusterable data set which implies that $|\mathcal{C}_j| = \Omega(\frac{N}{k})$. Thus, $|\mathcal{C}_{j,D}| = \Omega(\frac{N}{k^D})$. Note this assumption is crucial to remove the dependency with $\sqrt{N}$ that comes from the linear scaling with $\|Y\|$.

Repeating the above procedure for the rest of the leaf nodes leads to the total time complexity

$$\begin{aligned} T_{\text{leaf-label}} &= \mathcal{O}\left(\frac{T_\xi k^{\frac{3D}{2}}}{\epsilon_3}\right) \\ &= \mathcal{O}\left(poly\log(Nd)\frac{Dk^{\frac{5D}{2}}}{d}\log(k)\log(1/\Delta)\eta_1\epsilon_1\epsilon_3\right). \end{aligned} \tag{121}$$

□

I Numerical simulations

We use the k-means implementation in the library pyclustering [47]. This implementation contains a feature where the user can pass as an argument the distance metric to use. We initialized the centroids for k-means with k-means++ technique [4]. This is done at all the depths of the tree. For the Pearson correlation (or the point-biseral correlation), we use the r_regression function defined sklearn.feature_selection. We took the absolute values and then we normalized the vector. The maximum number of iterations taken for k-means algorithm for the cluster centroids convergence was set to 100. However, it was seen that for the datasets considered, the convergence was achieved with much less iterations. Note that we used the Euclidean distance for the distance calculation, as defined in Eq 1. For the regression task, we consider the dataset known as the Boston housing [24]. We did some feature selection so as the features used were eight ('LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE'). We also removed the skewness of the data through log transformation. Note that when we refer to the number of clusters (k) for a tree, we refer to the number of clusters that each node in the tree was split by the proposed supervised clustering method. In some cases, the samples could not be divided into the k desired number of clusters because of the number of samples. That is why in the Table 2, Table 3 and Table 4 the values in the tree size column may not be integer values.

Disclaimer

This paper was prepared for informational purposes by the Global Technology Applied Research center of JPMorgan Chase & Co. This paper is not a product of the Research Department of JPMorgan Chase & Co. or its affiliates. Neither JPMorgan Chase & Co. nor any of its affiliates makes any explicit

or implied representation or warranty and none of them accept any liability in connection with this paper, including, without limitation, with respect to the completeness, accuracy, or reliability of the information contained herein and the potential legal, compliance, tax, or accounting effects thereof. This document is not intended as investment research or investment advice, or as a recommendation, offer, or solicitation for the purchase or sale of any security, financial instrument, financial product, or service, or to be used in any way for evaluating the merits of participating in any transaction.