Executive Summary of the Thesis

A Survey on Feature Selection Methods for Classification Solved with Quantum Annealing

Laurea Magistrale in Computer Science & Engineering - Ingegneria Informatica

Author: Fabio Moroni

Advisor: Prof. Paolo Cremonesi

Co-advisor: Dott. Maurizio Ferrari Dacrema

Academic year: 2020-2021

1. Introduction

Machine Learning applications are nowadays getting more and more popular on a growing range of domains. Furthermore, the inner mathematical complexity and size of the problems are expanding, requesting more resources and execution time to be solved. Feature Selection is the process of selecting a subset of relevant features of a dataset, thus reducing the amount of data to use. However, it is computationally expensive, so finding more efficient strategies and approaches would lead to a significant impact. This thesis focuses on the filter category of Feature Selection methods for supervised learning classification problems, and explores the performances of some algorithms executed on a *Quantum Annealer*, which is a quantum computing machine that leverages specific quantum mechanics properties. This is a promising technology that offers scalability and speed-up potential for heavy and large computational tasks.
The goal of this work is to evaluate, through a series of experiments, a graph based approach, called *Graph Mutual Information QUBO*, *Quantum-Boosting* and *Quantum-Correlation* algorithms, which can all be executed on Quantum Annealing devices. These methods are compared between themselves and with classical non-quantum ones.

2. State of the art

This section briefly describes some classical and non-quantum feature selection methods used in the experiments. Then, a brief description of the Quantum device used is presented, along with *Quantum Annealing*, focusing on which kind of problems it can efficiently tackle.

2.1. Filter Feature Selection methods

This thesis focuses on filter Feature Selection methods and on classification problems, in which every sample is associated to different features. The goal of Feature Selection is to find the subset of features that can best describe the target variable to learn and improve the classification accuracy.
In this thesis, four different classic feature selection methods have been implemented and tested: *Variance Threshold*, which uses the Pearson correlation (Equation (1)), *Mutual Information*, based on entropy (Equation (2)) and mutual information (Equation (3)), *Chi2 Test* and *ANOVA F-Test* [2, 10]. In the referenced formulas, reported below, X and Z are both ran-

dom variables, respectively with n and m possible outcomes ($X_1, \dots, X_n$ and $Z_1, \dots, Z_m$).

$$Corr(X,Z) = \frac{Cov(X,Z)}{\sigma_X \, \sigma_Z} \tag{1}$$

where $Cov(X,Z)$ is the *covariance* between the random variables, and σ_X and σ_Z are the standard deviations of the random variables.

$$H(X) = -\sum_{i}^{n} p(X_i) \, log_2(p(X_i)) \tag{2}$$

$$MI(X;Z) = -\sum_{i}^{n}\sum_{j}^{m} p(X_i, Z_j) \, log_2 \frac{p(X_i|Z_j)}{p(X_i)} \tag{3}$$

where $p(X_i)$ is the probability of the i-th outcome of the random variable X, and $p(X_i|Z_j)$ is the conditional probability of outcome X_i given the outcome Z_j.
Generally, with this type of feature selection methods, a "score" is associated to every feature of the dataset and a ranking is generated. For instance, *Variance Threshold* and *Mutual Information* compute a score to each feature in different ways. *Variance Threshold* computes the Pearson correlation, defined with Equation (1), between the features f_i and the target variable y of the dataset (so, $Corr(f_i, y)$). *Mutual Information* assigns to each feature a score with Equation (4), named $Rel(\cdot,\cdot)$, which indicates the *relevance* between two variables.

$$Rel(f_i, y) = \frac{MI(f_i, y)}{H(f_i) + H(y)} \tag{4}$$

where:

- f_i is the i-th feature of the dataset
- y is the target variable of the dataset

The decision on which features to select follows two possible criteria: one way is to pick a desired number of features, k, with the highest score. The second one is to select all features which score is higher than a fixed threshold.

2.2. Classifiers

In this thesis, the *Random Forest* (RF) classifier [4] has been used for the experiments and results. The *Support Vector Machine* (SVM) classifier [3] has been used as well in one of the quantum-based feature selection algorithms described in the third chapter.

2.3. Graph theory overview

A graph $G(V,E)$ is a mathematical structure composed by a set of vertices V (nodes) and set of edges E. It can be *directed* or *undirected*, and *weighted* depending on the values associated to each edge. In our case, we will focus on undirected and weighted *complete* graphs, which means that each node is linked with every other node inside the graph. Different search problems involve graph theory, the one we are more interested in is *weighted MaxCut*. The weighted MaxCut problem consists in finding the subset of vertices S that maximizes the weight of the edges in the cut; that is, the sum of the weights of the edges with one endpoint in S and the other in its complement.

2.4. Quadratic Unconstrained Binary Optimization problems

Quadratic Unconstrained Binary Optimization (QUBO) consists in a class of NP-Hard problems that covers a wide range of applications in the combinatorial optimization domain. QUBO problems have the goal to minimize a *quadratic* objective function, with no constraints. QUBO problems are defined with binary variables, and they are mathematically defined as:

$$min \quad q(x) = x^T \, Q \, x \tag{5}$$
$$x \in \{0;1\}^n$$

where:

- x is the vector with n binary variables
- n is the dimensionality of the problem
- Q is a $n \times n$ symmetric (or upper-triangular) matrix describing the quadratic function $q(x)$

There are different meta-heuristics to solve this combinatorial problem. The one that we are more interested in is *Quantum Annealing*.

2.5. Quantum Computing

Quantum Computing refers to a specific branch of Computer science and Engineering where the core functionalities of the computers are based on quantum mechanics. The objective of this branch is to leverage and exploit the properties of quantum mechanics in order to overcome the current limits of classical computing, by embedding and using the quantum mechanics phenomena in a computational model.

There are several types of quantum computers. In this work we will focus on the *quantum annealer*, which is used to solve the presented quantum-based methods.

2.5.1. Quantum Computing fundamentals

With quantum computing there is a shift from the well known bit to its quantum counterpart, called *qubit*. While classic bits are discrete and deterministic variables whose values can be either 0 or 1, qubits are continuous and *probabilistic*, since a qubit can have a value in the range between 0 and 1. Furthermore, a qubit is a quantum object, thus it is capable of exploiting specific quantum characteristics and properties such as *superposition* and *entanglement*.

Superposition is the physical property of a quantum object to be in two different states "simultaneously". Despite this, at the end of the annealing procedure all the qubits collapse in one of the two classical states.

The **entanglement** property involves two or more qubits: it consists in a quantum phenomenon that creates a "bond" between the qubits. It has no correspondence in classical computing, it can be simulated, but inefficiently. When two qubits are entangled, their states can only make sense if observed and measured together, they cannot be described singularly. Thus, measuring one of them leads to the collapse into a classical state also of the entangled one.

The final collapsed classical state measured of the qubits depends on both superposition and entanglement.

2.5.2. Quantum Annealing

Quantum Annealing (QA) is a meta-heuristic used to determine the minimum energy *state* of a system. In our case, a *state* corresponds to the classical binary values (0 or 1) of the collapsed qubits of the *Quantum Processing Unit* (QPU), which is the hardware core of quantum annealers. QA simulates a quantum phenomenon called *Quantum Tunneling*. Quantum tunneling happens when a particle/wave-function can propagate through an high and "thin" energy barrier. This effect is one of the quantum capabilities that cannot be efficiently simulated through classical computing. In our case, the problem is represented as the energy of a system, and the objective is to find the solution that corresponds to the lowest energy, the global minimum of an energy function. Quantum tunneling is exploited in the quantum annealing process, thus speeding up the search for the global minimum. A classical and non-quantum approach, instead, would "climb" the energy hill, wasting more time to search for another potential solution. The quantum tunneling phenomenon is exploited to solve NP-Hard problems and obtain acceptable solutions, but with the possibility of leveraging the quantum properties of qubits in the attempt to reduce the needed computational time.

The energy of the each state is represented with its *Hamiltonian*, which is a mathematical description of the energy of a system (Equation (6)) obtained through the sum of the energies of its subparts, which in our case are the qubits and the interaction between themselves.

$$\mathcal{H} = \underbrace{\mathcal{H}_{\mathcal{I}}}_{\text{Initial Hamiltonian}} + \underbrace{\mathcal{H}_{P}}_{\text{Final Hamiltonian}} \tag{6}$$

$$\mathcal{H}_{\mathcal{I}} = -\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x^{(i)}\right)$$

$$\mathcal{H}_P = \frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i>j} J_{ij}\hat{\sigma}_z^{(i)}\hat{\sigma}_z^{(j)}\right)$$

where:

- $\mathcal{H}_{\mathcal{I}}$ is the *Initial Hamiltonian*
- $\mathcal{H}_P$ is the *Final* (or *Problem*) *Hamiltonian.* This term is determined by the biases h_i and the coupling strengths J_{ij}, which are set by the machine user.
- $A(s)$ and $B(s)$ functions can be considered as "weights" of $\mathcal{H}_{\mathcal{I}}$ and $\mathcal{H}_P$, they control the annealing process.
- s, the argument of $A(\cdot)$ and $B(\cdot)$, indicates the fraction of the annealing process, so it is a value between 0 and 1.
- $\hat{\sigma}_{x,z}$ are the Pauli matrices and represents the state of the qubits.
- h_i is the *bias* of the i-th qubit.
- J_{ij} is the *coupling strength* between the i-th and j-th qubit.

The bond with the QUBO formulation is evident: the *problem Hamiltonian* $\mathcal{H}_P$ coincides with a quadratic function that can be described with a $N \times N$ symmetric matrix, containing the biases h_i on the diagonal, while the coupling strengths J_{ij} correspond to the off-diagonal elements. This means that if we are able to create an optimization problem that can be mapped to $\mathcal{H}_P$, a solution can be obtained through Quantum Annealing. Once the QUBO formulation of the starting problem is obtained, ideally we would like to associate each logical variable of the problem to a qubit, but that is not always possible, due to the architecture of the QPU. The QPU consists in an arrangement of qubits that are connected between each other (with *couplers*, edges whose weight indicates the entanglement), thus creating a graph. This graph is not complete, most couplers are missing, and this raises an issue. In fact, going back to the mathematical model, the ideal scenario is to map each logical variable of the model to a single qubit, but as the dimensionality of the problem increases, this is no more possible. The square matrix that defines the QUBO objective function implies that there is a coupler (edge) for each pair of nodes, thus building a fully connected graph. As already seen, this is not the case. This issue is handled with a procedure called *minor embedding* [5]: the problem variables are duplicated and represented with multiple physical qubits, in order to make a 1:1 association between the variables of the QUBO model and the qubits themselves. This step will generate a *chain* of qubits that represent the same problem variable. In order to maintain consistency in the final results, all the qubits belonging to a chain must be equal. A chain *breaks* if the qubits belonging to it have different values, this implies that the solution obtained is inconsistent, and probably suboptimal.

Solving large problems with many variables is an issue with quantum annealers, since they are limited on the number of logical qubits embeddable onto the QPU. That's where Hybrid Quantum-Classical systems can help. This type of system decomposes the starting problem into subproblems that are solved using both classical and quantum computing.

3. Quantum-based algorithms

The general schema of the algorithms described in this chapter is the same:

1. define the formulas with statistical measures (e.g. mutual information, correlation and entropy) to build the matrix Q
2. create the matrix Q that describes the QUBO problem
3. embed the problem onto the QPU
4. solve the problem with the Quantum Annealer

Since we are dealing with the problem of feature selection, the final solution of the quantum annealing phase is an array of binary values, indicating which features are kept as relevant (1) or discarded (0).

3.1. Graph based method: Graph Mutual Information QUBO

Graph Mutual Information QUBO (Graph-MIQUBO) is a graph-based mutual information method, where the square matrix defining the objective function coincides with the adjacency (or weight) matrix of the corresponding mutual-information based complete and weighted graph [11]. Solving the QUBO problem coincides with computing a weighted MaxCut solution, using a quadratic objective function, as explained in [9] by Xuan Vinh Nguyen et al. The square matrix for the QUBO problem is computed with Equation (7).

$$Q = \begin{cases} Q_{ij} = \frac{MI(f_i,f_j)}{H(f_i)+H(f_j)} & if\ \ i \neq j \\ Q_{ij} = 0 & if\ \ i = j \end{cases} \tag{7}$$

where:

- $MI(\cdot,\cdot)$ is the mutual information, Equation (3).
- $H(\cdot)$ is the entropy, Equation (2).
- f_i coincides with the i-th feature of the dataset.

3.2. Quantum-Boosting

Quantum-Boosting (Q-Boosting) algorithm was proposed in 2008 by Neven et al. and lately revisited in 2012 [8]. This algorithm exploits the *boosting* technique, which consists in the composition of very basic and weak learners trained on few features in order to obtain a strong learner that uses only relevant features. The formulation of the square matrix to describe the QUBO problem is given in Equation (8).

$$Q = \begin{cases} Q_{ij} = Corr(h_i, h_j) & if\ \ i \neq j \\ Q_{ij} = \frac{S}{N^2} + \lambda - 2 * Corr(h_i, y) & if\ \ i = j \end{cases} \tag{8}$$

where:

- $Corr(\cdot,\cdot)$ is the Pearson Correlation, defined with Equation (1).
- h_i coincides with the estimate of a Support Vector Machine (SVM) classifier trained **only** with the i-th feature of the analysed dataset.
- y is the target variable.
- S is the number of samples inside the dataset.
- N is the number of features.
- λ is an hyper-parameter of the algorithm.

3.3. Quantum-Correlation

Quantum-Correlation (Q-Correlation), introduced by R. K. Nath et al. in 2008 [7], is a QUBO feature selection algorithm that computes the Pearson correlation (1) between the features and the target variable and between the features themselves. The formula to compute the square matrix is showed in Equation (9).

$$Q = \begin{cases} Q_{ij} = Corr(f_i, f_j) & if \;\; i \neq j \\ Q_{ij} = Corr(f_i, y) & if \;\; i = j \end{cases} \tag{9}$$

where:

- $Corr(\cdot,\cdot)$ is the Pearson Correlation, defined in Equation (1).
- f_i coincides with the i-th feature of the dataset.

4. Results

This section shows the results obtained with experiments performed on various classification datasets. Each dataset is first split into training and test subsets, which are respectively used to train a classifier and to test its generalization capability. All the datasets have been split with a 70% - 30% distribution of samples for training and testing. For all three quantum-based methods implemented, the steps of the experiments are the same. In fact, given the training set of a dataset: first, the QUBO objective function is computed. Then, several solutions are computed with the quantum annealer. For each one of the solutions found, which consists in a vector of binary variables indicating the features selected and discarded, a Random Forest classifier is instantiated. After collecting all these classifiers, the validation phase is performed for each one of them by computing a 5-fold Cross-Validation, thus obtaining a classification accuracy "*score*" (named *cross-validation score*) that is used to determine the best classifier. The classifier with the highest accuracy score is defined as the best one. Then, the final testing phase takes place, where a confusion matrix is computed over the test set in order to visualize the generalization capability of the best trained classifier itself. Two tables with results are reported: each row corresponds to a dataset, and the columns report the different feature selection methods implemented. Each method contains two separated columns labeled with N and CV_score: N indicates the number of features of the model used to train the best classifier and CV_score the accuracy score, that we have called also *cross-validation score*. In the columns with the methods solved with Quantum Annealing, it is reported in parenthesis the values of *chain break fractions*. This is a float value between 0 and 1 indicating the proportion of qubit chains, created during the embedding, that broke during the search for the state with minimum energy. For the last three datasets, the Hybrid quantum system has been used instead. With the hybrid approach we are not able to determine the number (or fraction) of chains that are broken during the execution. These are the tables reported, with the following references:

- Table 1: comparison across the QUBO methods, solved with Quantum Annealing (QA).
- Table 2: comparison of Graph-MIQUBO (solved with QA) against classical methods.

The presented quantum-based algorithms solved with quantum annealer machines are capable of obtaining promising results with respect to the classical ones: all three QUBO feature selection methods are able to discard a considerable amount of features. Graph-MIQUBO is more indicated for datasets with smaller dimensionality, with remarkable reduction of the features used in the classification problem. Q-Boosting and Q-Correlation, on the other hand, are more indicated for problems with a larger set of features. With respect to other meta-heuristics, like *Simulated Annealing* [6] and *Steepest Descent* (a greedy local search approach [1]), Quantum Annealing is able to find a solution with a speed-up. For instance, with the "isolet" dataset that has

	All Features		Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score
breast_cancer	30	0.957	**17**	**0.957 (0.0)**	25	0.960 (0.0)	28	0.960 (0.0)
robot-failures-lp5	90	0.702	70	0.754 (0.48)	80	0.745 (0.31)	**19**	**0.754 (0.39)**
SPECTF	44	0.790	9	0.817 (0.23)	**12**	**0.838 (0.32)**	36	0.817 (0.02)
thyroid-ann	21	0.995	12	0.997 (0.0)	**11**	**0.997 (0.0)**	5	0.946 (0.0)
isolet	617	0.979	539	0.986	**114**	**0.986**	174	0.986
swarm-behaviour	2400	1.000	746	1.000	**171**	**1.000**	257	1.0

Table 1: Quantum Annealing approach results across the algorithms. In the *cross-validation score* (CV_score) field of the method columns, is reported in parenthesis the *chain break fraction*, except for the last two datasets, which are solved with the hybrid approach. Highlighted in bold the results where a QUBO feature selection method provided the highest number of features discarded and with the highest (or equal) score.

			Quantum Annealing		Classic Methods							
	All Features		**Graph-MIQUBO**		VarThr		MI		Chi2		ANOVA	
Dataset	N	Score	N	Score	N	Score	N	Score	N	Score	N	Score
breast_cancer	30	0.957	17	0.957 (0.0)	29	0.957	21	0.960	22	0.960	24	0.960
robot-failures-lp5	90	0.702	70	0.754 (0.48)	50	0.745	80	0.763	83	0.746	78	0.737
SPECTF	44	0.790	9	0.817 (0.23)	43	0.806	21	0.833	8	0.828	23	0.822
thyroid-ann	21	0.995	12	0.997 (0.0)	20	0.996	19	0.997	11	0.997	17	0.997
isolet	617	0.979	539	0.986	50	0.983	63	0.986	51	0.988	51	0.986
swarm-behaviour	2400	1.000	746	1.000	50	1.000	1811	1.000	1175	1.000	294	1.000

Table 2: Results of Graph-MIQUBO solved with the QPU compared against classical filter feature selection algorithms. In the *cross-validation score* field (here labelled as "Score") of the QUBO method column, is reported in parenthesis the *chain break fraction*, except for the last two datasets, which are solved with the hybrid approach.

617 features, these are the timings needed to find the optimal solution of QUBO formulated with Graph-MIQUBO algorithm:

- Quantum Annealing (hybrid): 1m 40s
- Steepest Descent: 20m 49s
- Simulated Annealing: 15h 21m 31s

The major drawback and bottleneck of QUBO feature selection methods is the time needed to generate the QUBO model itself. This becomes more evident by comparing a QUBO method with classical ones, which do not need any QUBO generation. For example, with "swarm-behaviour" dataset that has 2400 features, there is a striking total difference between Graph-MIQUBO (total time of 9 hours) and Variance Threshold (less than a second).

5. Conclusions

In this thesis, we presented three methods based on a QUBO problem solved with different meta-heuristics, including Quantum Annealing. These methods are: *Graph Mutual Information QUBO*, *Quantum-Boosting* and *Quantum-Correlation.* These algorithms have been tested and compared between themselves and with classical ones.
The results obtained are promising, considering that quantum annealing devices is relatively new technology. The next steps for this field of research is to improve and speed-up the time needed to compute a QUBO formulation, and to express deeper insights of interaction and relevance between the features themselves and with the target variable. Furthermore, a development of quantum annealers' technology could open up new possibilities for larger experiments that can be carried out without any hybrid approach.

References

[1] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization.* Princeton University

Press, 2003.

[2] Langley P. Blum A. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, page 245, 1997.

[3] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2006.

[5] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, Oct 2008.

[6] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983.

[7] Rajdeep Kumar Nath, Himanshu Thapliyal, and Travis S. Humble. Quantum annealing for automated feature selection in stress detection. 2021.

[8] H. Neven, V.S. Denchev, Geordie Rose, and William Macready. Qboost: Large scale classifier training with adiabatic quantum optimization. *Journal of Machine Learning Research*, 25:333–348, 01 2012.

[9] Xuan Vinh Nguyen, Jeffrey Chan, Simone Romano, and James Bailey. Effective global approaches for mutual information based feature selection. page 512–521, 2014.

[10] Debbie L Hahs-Vaughn Richard G Lomax. *Statistical concepts: a second course*. 2013.

[11] Zhihong Zhang and Edwin Hancock. A graph-based approach to feature selection. 05 2011.

Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

A Survey on Feature Selection Methods for Classification Solved with Quantum Annealing

Supervisor: Prof. Paolo Cremonesi
Co-supervisor: Dott. Maurizio Ferrari Dacrema

Master thesis by:
Fabio Moroni, mat. 946779

Academic Year 2020 - 2021

Abstract

Modern Machine Learning problems are getting more and more complex: their size is increasing and the challenges to face are harder to overcome. The growth of the datasets implies an huge increase of the computational time to execute algorithms and train learners. Solving Feature Selection problem reduces the amount of data used to learn a process, but it is computationally expensive. Thus, finding more efficient strategies and approaches would lead to a significant impact.

Feature Selection is a phase of a Machine Learning pipeline that has the goal to reduce the number of features of a dataset, without loosing generality and accuracy of the learner used. Solving efficiently this problem is crucial: we may be able to significantly reduce the number of features of a dataset, which implies a speed up of the training phase of the learner. Despite this, the accuracy obtained should be equal or better in both validation and test steps of the pipeline, thus improving its generalization ability. Furthermore, in a research scenario, cutting unnecessary or less relevant features gives a deeper insight about the dynamics of the process to learn. However, solving a feature selection problem can be computationally very expensive. So, it needs some speed-ups in terms of resources and new technologies.

This thesis focuses on the filter category of Feature Selection methods for supervised learning classification problems. These techniques exploit statistical measurements (e.g. mutual information or correlation) between the features themselves and with the target variable(s) in order to determine their relevance for the problem.

The goal of this work is to evaluate a graph based mutual information approach, called *Graph Mutual Information QUBO*, *Quantum-Boosting* and *Quantum-Correlation* methods, which can all be executed on Quantum Annealer devices. The evaluation is achieved through a series of experiments, which results are used to compare the quantum-based algorithms to classical (non-Quantum) ones from the point of view of accuracy, efficiency, and execution timings.

Sommario

I problemi moderni di Machine Learning stanno diventando sempre più complessi: la loro dimensionalità continua a crescere a le sfide da affrontare sono sempre più difficili. Questa crescita riguarda i set di dati impiegati, la quale implica un grande aumento del tempo computazionale necessario per eseguire gli algoritmi e poter "istruire" i regressori o classificatori. Una soluzione a questo problema è una efficiente esecuzione della fase di *Feature Selection*, letteralmente *selezione delle caratteristiche.*

Feature Selection è una delle fasi di un algoritmo di Machine Learning che ha l'obiettivo di ridurre il numero di *features* (appunto, *caratteristiche*), senza però influenzare in maniera negativa la precisione e accuratezza del regressore o classificatore (*learner*), impiegato per imparare le dinamiche di un processo. Risolvere in maniera efficiente questo problema risulta estremamente importante, poichè diminuendo potenzialmente gran parte delle *feature* di un set di dati si ottiene una riduzione signicativa della complessità di un problema, nonchè del tempo necessario per l'addestramento del *learner.* Nonostante questa simplificazione, le performance dei nuovi modelli possono rimanere uguali oppure anche migliorare sia negli step di validazione che di test. Inoltre, soprattutto in analisi e problemi di ricerca, riuscire ad eliminare *feature* che risultano non necessarie e meno rilevanti può contribuire ad una maggiore comprensione delle dinamiche del processo da studiare.

Tuttavia, risolvere il problema di *Feature Selection* può essere molto dispendioso dal punto di vista computazionale, con tempi di esecuzione che non rendono utile l'impiego e la soluzione di tale fase. Per questo motivo, è necessario migliorare le tecniche impiegate, nonchè le risorse e sfruttare nuove tecnologie.

Questa tesi si concentra sui metodi di selezione di *feature* di tipo "filtro", chiamati *filter methods*, e su problemi di classificazione, con a disposizione anche la variabile *target* da imparare (*supervised classification problems*). I *filter methods* sfruttano delle grandezze statistiche (come *mutual information* e correlazione) tra le features stesse e con la variabile target, in modo tale da determinare la loro rilevanza per il problema.

L'obiettivo di questa tesi è quello di testare e analizzare un metodo basato sui grafi e la *mutual information*, chiamato *Graph Mutual Information QUBO*, e di valutare i metodi *Quantum-Boosting* e *Quantum-Correlation*, i quali possono essere tutti eseguiti con un *Quantum Annealer*. Infine, confronteremo questi algoritmi con quelli classici (di tipo non-Quantum) dal punto di vista dell'accuratezza, efficienza e tempi di esecuzione.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine Learning applications are nowadays getting more and more popular on a growing scale of domains. Furthermore, the inner mathematical complexity and size of the problems are expanding, requesting more resources and execution time to be accomplished. To handle these issues, solving efficiently the problem of Feature Selection becomes crucial.

Feature Selection (*FS*) consists in reducing the number of features of a dataset, without loosing generality and accuracy of the classifier (or regressor) used to learn a concrete process. This phase of the pipeline is important because we may able to solve the mentioned issues: significantly reduce the size of the problem, allow for a faster execution of the training phase of the classifier without loss of generality and also grasp deeper information about the dynamics of the problem. Indeed, imagine a medical problem of cancer classification or detection: in a study or research phase, given a dataset with an excessive amount of features, a feature selection algorithm could help not only to improve the training time of specific classifiers, but also to better understand the main correlation between the causing factors and formation of the cancer itself.

Out of all the techniques and categories of approaches to solve Feature Selection, in this work we will focus only on supervised learning classification problems with *filter methods*, which consist in algorithms that exploit statistical measurements (e.g. correlation) between the features themselves and with the target variable(s) to determine their relevance for the problem.
The main statistical measures used for this purpose are the Mutual Information (MI), which consists in a dependence measure between two random variables obtained through their entropies, and the correlation between two variables.
In our work these variables coincide with the features of a given dataset, thus per-

forming feature selection by considering the "most relevant features" to solve classification problems.
The first approach described in this thesis uses Mutual Information: it consists in an algorithm that transforms the FS problem into a weighted MaxCut Graph-based combinatorial optimization problem. Since this problem has to be adapted to a QUBO formulation in order to be exploited with a Quantum Annealer, this approach is called *Graph Mutual Information QUBO* (Graph-MIQUBO). Two other quantum-based approaches are explained and tested in this work: *Q-Boosting* and *Q-Correlation*. The former leverages the idea of boosting in Machine Learning, which means to combine several simple classifiers to obtain a stronger and more robust one. *Q-Correlation* formulates an optimization problem by using the *Pearson correlation* between the features themselves and with the target variable.

The main goal of this thesis is to evaluate three feature selection methods that can be executed on Quantum Annealer devices: a graph based mutual information approach, called *Graph-MIQUBO*, Q-Boosting and Q-Correlation. Then, we report the results of our experiments, analysing their overall performance from both accuracy and efficiency point of view with respect to classical and non-Quantum algorithms.

Chapter 2

State of the art

Feature Selection is a crucial phase in the Machine Learning pipeline, where the main goal is to reduce the number of features, such that the generalization power of the predictor and model used is not negatively affected, but it may also get enhanced. This step is getting more and more important in modern problems, since they become more complex and their mathematical dimensionality gets too large too handle them in reasonable execution times.

In this chapter, we are going to describe in general the overall problem of Feature Selection and what are the different categories of methods. Then, we are going to focus on one of these categories (the *filter methods*) and discuss various techniques of this type that have been tested.

At the end of this chapter, a brief description of the Quantum device used is made, along with *quantum annealing*, focusing on which kind of problems it can efficiently solve.

2.1 Overview of Feature Selection methods

When dealing with Feature Selection, one possible way to determine the best subset of features is to use a *brute-force* approach: use all the possible combinations of feature subsets, train a classifier with it and evaluate the performance obtained. It is trivial to understand that when the dimensionality of the problem (e.g. the number of features) increases then the cardinality of all the possible combinations to test grows significantly, thus making the computational cost hugely expensive.

Since the *brute-force* approach becomes unfeasible as the number of feature grows, there are other techniques that are used for this purpose that can be divided in three main categories:

- **Filter** methods: the subset of features is selected independently from the data modelling used and it is based primarily on statistical measures.
- **Embedded** methods: these are called also *hybrid* methods, since the selection of the subset of features occurs during the execution of the modelling algorithm. Some of these methods, like Lasso or Ridge Regression, perform feature weighting and can be applied to any type of learner (a Support Vector Machine with linear kernel, for instance).
- **Wrapper** methods: feature subsets are selected based on the performance on a modelling and learning algorithm, considered as a black box.

All these methods share the same objective: find the best possible subset of features that maximizes the generalization power of the learner (no matter if it is a classification, regression or forecasting problem). [1, 14]

In this work the focus is on classification problems with *filter* methods, which are going to be described in details in the next section.

2.2 Filter methods for Feature Selection

Filter methods for Feature Selection have the capability of finding the best subset of features by computing statistical measures [46] between the features themselves and with the target variable that has to be learned. Every method applies its own criteria and measures to compute the *relevance* of a feature [7] with respect to the other features and for the target variable. Generally, with this type of methods, a "score" is associated to every feature of the dataset and a ranking is generated; thus, through this ranking, it is possible to determine the most relevant features that best describe the problem.

2.2.1 Variance Threshold

The *Variance Threshold* method (or *Correlation criteria* [1, 14]) is one of the simplest algorithms to use in order to rank the features of a dataset [47, 18, 6, 19]. The statistical measure considered is the *Pearson correlation* [3] between the single features and the target, following this formula:

$$Corr(f_i, y) = \frac{Cov(f_i, y)}{\sigma_{f_i}\, \sigma_y} \tag{2.1}$$

where:

- f_i is the i-th feature (column) of a dataset
- y is the target variable of the learning problem
- $Cov(f_i, y)$ is the covariance between the two variables
- σ_{f_i} is the standard deviation of the i-th feature
- σ_y is the standard deviation of the target variable

Thus, this method ranks the features starting with the ones that are more *correlated* with the target variable, without taking into consideration the *interactions* between the features. There are two criteria in order to decide which features to use before the final test evaluation with a classifier: one way is to pick the top k features of the ranking, where k is an hyper-parameter to fix a priori. The second one is based on selecting the features which ranking values are higher with respect to a fixed threshold.

2.2.2 Mutual Information based methods

There are plenty of different filter feature selection methods that use at their base the statistical measure of *Mutual Information*, also known as *Shannon's Information* [55, 44, 54]. Mutual Information is computed using the *entropy*, which measures the level of "uncertainty" of a random variable. Given a random variable X, with n possible outcomes $x_1, \ldots, x_n$, the entropy of X is computed in this way:

$$H(X) = -\sum_{i}^{n} p(x_i)\, log_2(p(x_i)) \tag{2.2}$$

The base of the *log* is 2 since the unit of the entropy is the bit. Given two random variables X and Y, respectively with n and m possible outcomes ($x_1, \ldots, x_n$ and $y_1, \ldots, y_m$), it is also possible to define the *joint entropy*:

$$H(X, Y) = -\sum_{i}^{n}\sum_{j}^{m} p(x_i, y_j)\, log_2(p(x_i, y_j)) \tag{2.3}$$

and the *conditional entropy*:

$$H(X|Y) = H(X, Y) - H(Y) = -\sum_{i}^{n}\sum_{j}^{m} p(x_i, y_j)\, log_2(p(x_i|y_j)) \tag{2.4}$$

With these (2.2), (2.3) and (2.4) formulas, it is possible to compute the *Mutual Information* (MI) between two random variables X and Y, that is a non-negative value that indicates how these are related:

$$MI(X;Y) = -\sum_{i}^{n}\sum_{j}^{m} p(x_i, y_j)\, log_2 \frac{p(x_i|y_j)}{p(x_i)} \tag{2.5}$$

The higher the value of the Mutual Information, the higher is the certainty of the relation between the variables themselves.

Thus, similar to the previous method, it is possible to formulate a ranking of the features of a dataset by computing $MI(X_i, y)$, where y is the target variable. Secondly, choose a value for the hyper-parameter k and then pick the top k features from the computed ranking.

As indicated in [55], through Mutual Information it is possible to describe the concept of *interaction* between variables, since $MI(X, Y)$ can be seen as a measure of the strength of a 2-way interaction between X and Y. Furthermore, as described in [56], with Mutual Information we introduce with a formula the concept of *relevance* of a variable with respect to another one:

$$Rel(X;Y) = \frac{MI(X;Y)}{H(X) + H(Y)} \tag{2.6}$$

Namely, the ratio between the Mutual Information of X and Y over the sum of the single entropies.

By considering the relations between the Mutual Information and the entropy [55], it is possible to rewrite the formula of the *relevance* only in terms of entropies. In fact, given:

$$MI(X;Y) = H(X) + H(Y) - H(X, Y)$$
$$Rel(X;Y) = \frac{MI(X;Y)}{H(X) + H(Y)}$$

we obtain:

$$Rel(X;Y) = \frac{H(X) + H(Y) - H(X, Y)}{H(X) + H(Y)} = 1 - \frac{H(X, Y)}{H(X) + H(Y)} \tag{2.7}$$

It is clear that $Rel(X;Y)$ is a value between 0 and 1: the closer to 1 it is, the higher is the relevance of X to better "understand" Y; that is the case because this scenario happens when $H(X, Y)$ is close to zero, which means that the "uncertainty" of X and Y jointly is extremely low.

2.2.3 Chi2 Test

The *Chi2 Test*, which is called after the *Chi-square* statistic, is another filter method that can be used for feature selection. Similar to the Variance Threshold method seen in section 2.2.1, the underlying idea is to compute a statistic measure, thus giving a score and a ranking to the features of a dataset. Given a random variable X, with n possible outcomes $x_1, \ldots, x_n$, the Chi-square statistic value is computed, as explained in [34, 53]:

$$\chi^2 = \sum_i \frac{(x_i - \overline{x}_i)^2}{\overline{x}_i} \tag{2.8}$$

where:

- x_i : is the observed value of random variable X
- $\overline{x}_i$: is the expected value of random variable X if the null-hypothesis is true

The Chi Squared test measures dependence between stochastic variables, thus showing which ones are less "relevant"; in our case, by performing the chi-square test between features and the target variable, the features with the lowest score are the ones that provide less information to learn the correct samples classification.

2.2.4 ANOVA F-Test

The name *ANOVA* comes from Analysis of Variance [47, 18, 6], while *F-test* is named after the mathematician and statistician Fisher, who introduced the F value, computed as [28]:

$$F = \frac{MS_B}{MS_W} \tag{2.9}$$

where:

$$MS_B = \frac{\sum_i^m n_i\,(\overline{x}_i - \overline{x})^2}{m-1} \tag{2.10}$$

$$MS_W = \frac{\sum_i j(x_{ij} - \overline{x}_i)^2}{n-m} \tag{2.11}$$

As described in [28], the terms of the above formulas are:

- m is a prefixed value indicating the number of "groups" to compare: in our case with classification problems, m indicates the number of classes (in binary problems $m = 2$, and so on)

- MS_B is the "*between*-group variability"
- MS_W is the "*within*-group variability"
- n is the total number of observations, while n_i only for the i-th group
- $\overline{x}_i$ denotes the empirical mean of the samples in the i-th group, while $\overline{x}$ denotes the overall mean of all the observations
- x_{ij} denotes the j-th observation in the i-th group

The *ANOVA F-Test* feature selection method minimizes false negative errors and, once again, associates to each feature of a dataset a score, thus producing a ranking from which it is possible to pick the top k features.

2.3 Classifiers

In this thesis, as already mentioned, we focus on supervised classification problems, thus the "learner" is a *classifier*, which will output a *label* corresponding to the estimated learned class of the target variable.

There are several classifier models and methods to train them efficiently, here we describe the *Random Forest* (RF) classifier, which is the one that has been used for the experiments and results, and also the *Support Vector Machine* (SVM) because it is used in one of the feature selection algorithms discussed in the third chapter.

2.3.1 Random Forest classifier

Random Forest (or random decision forest) is an ensemble learning method that can be used for both classification and regression problems that leverages the usage of multiple decision trees at training time. It is very powerful and flexible to be applied also on large-scale datasets. Furthermore, it is a continuous object of research for its versatility on a growing scale of problem domains, due to its inner relevant mathematical and statistical mechanisms, such as the selection of parameters, the resampling steps, and variable importance measures [5]. Since we consider only classification problems, we describe the behaviour of this learner as a classifier.

This method was first introduced by Tim Kam Ho in 1995 [30]; nearly ten years later in 2006, Leo Breiman developed the algorithm [10] and registered "Random Forest" as a trademark.

In classification scenarios, a random forest classifier generates a series of decision trees each on different and random subsets of the available features of a dataset. This procedure is called ”feature bagging”, referring to the *bagging* ensemble technique, because if one or more features are highly relevant to correctly predict the target variable, the generated decision trees that use those relevant features will be more ”frequent” and thus the probability of making a correct prediction increases. This would make these ”strong” trees and features to be correlated, leading to an increase of the overall and final accuracy of the random forest classifier.

Once all these decision trees perform a prediction of a target class, it is counted; the output of the random forest is just the single label of a target class that has won with majority voting.

2.3.2 Support Vector Machine classifier

Support Vector Machine (SVM) is a supervised learning model first introduced ad developed in the 90's [9, 16], and can be used for different machine learning task, such as regression, classification and clustering. For classification tasks, the objective of this model is to find a subset of samples, called *support vectors*, which are used to determine the maximum *margin*.

Consider a binary classification problem, so a problem with a target variable that contains two different classes, the margin corresponds to the distance between the points (e.g. samples) belonging to the different classes (Figure 2.1). Thus, training an SVM consists in solving a maximization problem where the objective function describes the margin itself. The samples that locate this margin are called support vectors.

Support Vector Machines are capable to solve also multi-class problems: for example, is to use different binary SVM classifiers that distinguish between one of the labels and the rest (*one-versus-all*) or between every pair of classes (*one-versus-one*).

This classifier model has been and still is one of the most popular, due to its inner structure and training process and also for its capability to determine also non-linear boundaries between different classes [9]. It has been successfully used in different applications and domains, for example in biology [11] and also text categorization [31].

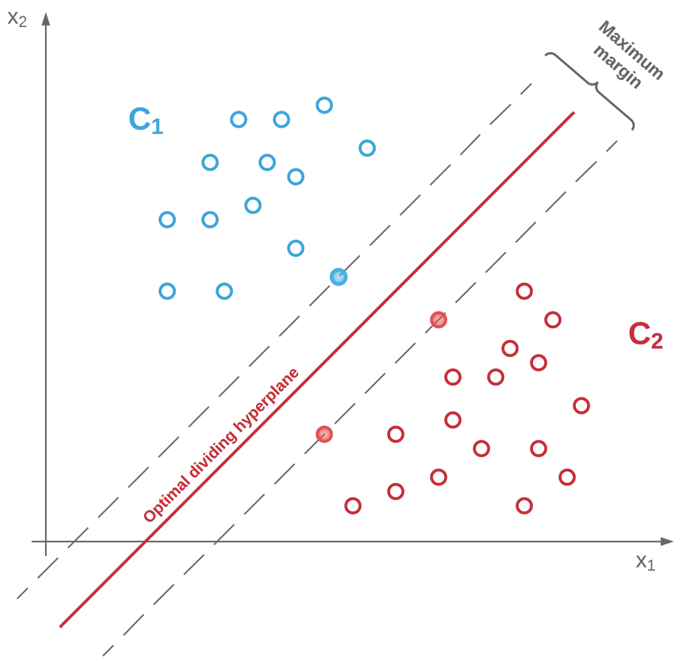


Figure 2.1: SVM margin example on binary classification problem.

2.4 Graph theory overview

This section introduces the main concepts and definitions about graph theory, since they are going to be used throughout the description of our model, detailed in the third chapter.

A graph G is a mathematical structure composed by nodes and edges; it is used as a base model to represent and describe different problems and processes, such as social network interactions [13], transportation [45], tasks assignment logic [50], biomolecular structure [36] and many more [21].

The graph is fully described as $G(V, E)$, where V is the set of vertices and E the set of edges and it can be *directed* or *undirected*, *weighted* or not depending on the values associated to each edge.

For our case, we will focus on undirected and weighted *complete* graphs, which means that each node is linked with every other node inside the graph.

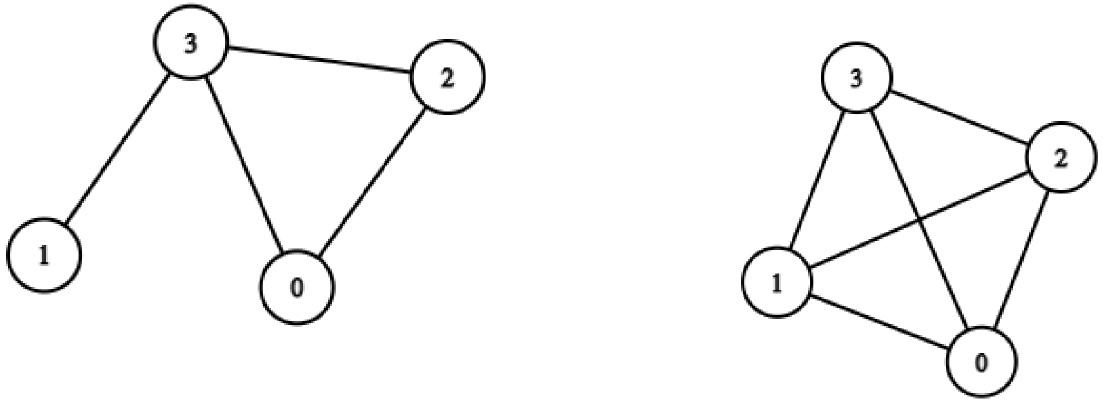


Figure 2.2: Non-complete vs complete graphs

2.4.1 Weighted MaxCut problem

A graph is described through an *adjancency* (or *weight*) matrix W, that is a $|V| \times |V|$ symmetric matrix with 0s on the diagonal and the weights of the linking edges as off-diagonal elements. If there is no edge between two nodes, a 0 is placed instead. Notice that, since we are focusing on complete undirected graphs, the weight matrices that are going to be used will contain zeros only on the diagonal and are symmetric.

The weighted MaxCut problem, described by R. M. Karp in 1972 [32], consists in finding the subset of vertices S that maximizes the weight of the edges in the cut; that is, the sum of the weights of the edges with one endpoint in S and the other in its complement.

This problem is can be solved with a variety of methods and has been a hot topic of study and analysis in the last decades [26], since it is an NP-Complete problem. But, for our study, we highlight the fact that the weighted MaxCut problem can be written as a binary maximization problem, by simply using the adjacency matrix W to define a quadratic objective function:

$$\max_{x^n} x^T W x \tag{2.12}$$

$$x \in \{0,1\}^n$$

This particular formulation is very helpful for our scope and objective, since it can be easily translated into a Quadratic Unconstrained Binary Optimization (QUBO), described in the following section.

2.5 Quadratic Unconstrained Binary Optimization problems

In this section, the mathematical fundamentals of the *Quadratic Unconstrained Binary Optimization* (QUBO) problems are described, since they are at the base of the Graph-MIQUBO algorithm which is going to be addressed in details in Chapter 3 of this work.

2.5.1 QUBO: Mathematical description

Quadratic Unconstrained Binary Optimization (QUBO) consists in a class of NP-Hard problems that covers a wide range of applications in the combinatorial optimization domain.

QUBO problems have the goal to minimize an objective function, which is a *quadratic* function, with no constraints on the search space of the solution. This solution has to be with binary variables, which means they can only assume two possible discrete values: 0 or 1.

A QUBO problem is mathematically defined as:

$$\begin{gathered} min \;\; q(x) = x^T \, Q \, x \\ x \in \{0; 1\}^n \end{gathered} \tag{2.13}$$

where:

- x is the vector with n binary variables
- n is the dimensionality of the problem
- Q is a $n \times n$ symmetric matrix describing the quadratic function $q(x)$

The quadratic objective function $q(x)$ can be rewritten in non-matricial form, thus obtaining the following equivalent problem formulation:

$$min \;\; q(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i \, x_j \, Q_{ij} \tag{2.14}$$

$$x \in \{0;1\} \quad \forall i,j = 0,1,...,n-1$$

For QUBO problems, the Q matrix is symmetric or upper triangular. It is possible to convert any square matrix into a symmetrical or upper triangular one without loss of information. By having an $n \times n$ matrix M, its symmetric equal Q is obtained in this way:

$$Q_{ij} = \frac{M_{ij} + M_{ji}}{2} \qquad \forall_{i,j} \; i \neq j \tag{2.15}$$

or in matricial form:

$$Q = \frac{1}{2}(M + M^T) \tag{2.16}$$

To obtain the upper triangular form:

$$\begin{aligned} Q_{ij} &= M_{ij} + M_{ji} & \forall_{i,j} \; j > i \\ Q_{ij} &= 0 & \forall_{i,j} \; j < i \end{aligned} \tag{2.17}$$

There's a wide range of applications for QUBO problems [25], for instance:

- Assignment problems: knapsack, portfolio, etc.
- Graph based analysis: MaxCut/MaxClique, partitioning and clustering.

2.5.2 Classical algorithms to solve QUBO

As stated previously, QUBO is a class of NP-Hard problems. Thus, in order to find a solution in a reasonable time, approximations and heuristics are needed.

In this specific subsection we are going to address some of these heuristics, which are going to be used for the experiment and results section of this work.

Simulated Annealing

Simulated Annealing (SA) is a metaheuristic algorithm used in order to approximate the global optimum of a given objective function. This algorithm is often used when the search space is discrete. The name *annealing* derives by metallurgy, because it consists in a technique involving heating and controlled cooling of a material in order to alter its physical properties: the atoms of the material re-arrange as the temperature of the system changes and gets low. The current arrangement of the atoms is called **state** and an **energy** is associated to it. As the temperature gets lower and lower, the atoms will arrange in a *stable* position, which coincides with the state at the *lowest energy*, defined *ground state*.

The algorithm is the one described in [33] by Kirkpatrick et al., and to make a proper association with the QUBO problem that we are starting from:

- the physical/atomic *state* corresponds to the current solution x_k of the algorithm
- the *energy* coincides with the objective function, in our case with the quadratic $q(x) = x^T\, Q\, x$
- the *ground state* corresponds to the global minimum and solution of the problem (2.13).

The algorithm starts with a random state, a fixed number of iterations to perform, and a high temperature value. Each iteration is associated with a temporary state. In each iteration the temperature is lowered, following a decreasing function determined a priori, and a random neighbor state of the current one is selected. With a certain probability that depends on the temperature, and on the energies of the current state and its selected neighbor, the neighbor state is selected to be the new one associated to the next iteration. This process lasts until the number of iterations are finished, and the state associated to the final one is the estimated global minimum of the energy function.
The fact that during its steps a random neighbor state of the current one is selected is an advantage of this approach: the neighbor state has a chance of having a higher (so, worst) energy. Thus, this feature increases the probability to escape from local optima and search for the global one.

To summarize the Simulated Annealing approach: the goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy, coinciding with an approximation of the global minimum of the objective function.

Tabu Search

Tabu Search (TS) is another metaheuristic search method proposed and refined by Glover in late '80s [23, 24]; it uses local search in order to find a global optimum in an optimization problem. It is similar to *Simulated Annealing*, since both methods search a possible solution "down hill" by testing neighboring states of the given one. Like SA, TS has the capability to accept with certain probability a state with higher energy, thus "climbing the hill". This behaviour is for both algorithms a strong one, since it is used to escape from possible local optima and search for the global one.

However, TS operates by considering also a *"Tabu List"*, which is a list containing all the previous explored moves that will no longer be considered; this is also the reason of the name of this method.

This "Tabu List" has a fixed length of n, an hyper-parameter of the algorithm. Thus, this data structure can be seen as a short-term memory containing solutions (in this case, states) to be avoided in the local search for the current iteration of the algorithm.

Steepest Descent

Steepest Descent is a technique that is used to find the global optimum of an objective function by following iteratively a specific direction. For QUBO problems the direction at each step is obtained by computing a *local* minimization, so with *local search* [2]. This computation consists in determining the dimension along which, after a variable flip (0 to 1 or viceversa), the highest energy drop is observed.

This algorithm is considered *greedy*, because the one direction or choice taken is based on a fixed criteria, in this case the one with the biggest energy drop. This approach has advantages as well as disadvantages: while this criteria drastically speeds up the search for an optimal solution, it has the main drawback of getting stuck into local optima, thus returning a solution that is not at the lowest possible energy.

2.6 Quantum Computing

In the last decades we have seen an increase in the amount of power for classical computers, speeding up processes, opening many branches of research and solving complex problems of any type: from economics, finance and logistics to bioinformatics and medicine.

Despite this evolution, there are still major limits with classical computing, especially in solving highly computationally expensive problems, like simulations, queries and heavy calculus tasks. These limits are linked with the core functionalities and properties of classical computers, and that's where Quantum Computing can help us, by leveraging its different computational paradigm.

Quantum Computing refers to a specific branch of Computer science and Engineering where the core functionality and "rules" of the computers are based on quantum mechanics. The objective of this branch is to leverage and exploit the properties of quantum mechanics in order to overcome the current limits of classical computing, by embedding and using the quantum mechanics phenomena in a computational model.

The idea and concept of quantum computers was first introduced by the physicist Paul Benioff in 1980 when he proposed a quantum mechanical model of the Turing machine [4]. This concept developed right after in 1982, when Richard Feynman and Yuri Manin suggested that classical computer could not simulate certain quantum mechanical effects [20, 35], thus creating the idea of quantum superiority regarding computational power.

The interest for quantum computing started to grow more and more in the 90s, because through applications and demonstrations a remarkable speed-up have been achieved with algorithms proposed by Peter Shor in 1994 [51, 52] and by Lov Glover in 1996 [27]: the former regarding integer factorization and the latter about the search of an element in an unordered list, both obtaining the wanted outputs in polynomial time.

After these discoveries, Quantum computing became increasingly active and central for researchers and witnessed many studies and developed technologies, specifically in the last decade [29].

In Quantum computing, there are several types of machines called *quantum computing systems*, and the two most relevant are *gate-based* quantum computing and *quantum annealing*, also referred to as *adiabatic* quantum computing.

In this work we will focus on *adiabatic* quantum computing, since it is the model that is used as base to develop the proposed algorithm and, of course, in the experiments as well.

2.6.1 Quantum computing fundamentals

Before entering in details about the quantum annealing model, we need to introduce the fundamental concepts and definitions of quantum computing and how quantum mechanics plays a role in all of this.

First of all, from classical computing there's a shift from the well known bit to its quantum counterpart, called *qubit*. While classic bits are discrete and deterministic variables whose values can be either 0 or 1, qubits are continuous and **probabilistic**, since a qubit can have a value in the range between 0 and 1.

Furthermore, a qubit is a quantum object, thus it is capable of exploiting specific quantum characteristics and properties such as *superposition* and *entanglement*. **Superposition** is the physical property of a quantum object to be in two different states "simultaneously"; in the case of the qubit, it is both 0 and 1 with different probabilities. Despite this, at the end of the annealing procedure all the qubits collapse in one of the two classical states.

Thus, the final collapsed state measured depends on the probabilities linked with the superposition just before the measurement itself, and it is also influenced by how the qubits are correlated and the coupling strengths applied to them.

While superposition is a property of a single qubit, *entanglement* involves two or more qubits: it consists in a quantum phenomenon that indicates a "bond" between the qubits; it has no correspondence in classical computing, it can be simulated, but inefficiently.

When two qubits are entangled, their states can only make sense if observed and measured together, they cannot be described singularly; thus, measuring one of them leads to the collapse into a classical state also of the entangled one.

2.6.2 Quantum Annealing

Given the definitions of the main quantum properties of *superposition* and *entanglement*, we can discuss in more details the quantum annealing system. The term "annealing" have been already introduced in Section 2.5.2 and, indeed, this model adopts the concepts of state and energy.

In fact, this type of machine exploits the natural behaviour of a system to reach its ground state, which corresponds to the one with the lowest energy, like it has been described with Simulated Annealing. However, the annealing process on a quantum annealer works in a different way: because it leverages a specific quantum property called *quantum tunneling* and no temperatures are used to control it.

Quantum tunneling happens when a particle/wave-function can propagate through an high and "thin" energy barrier and this effect is one of the quantum capabilities that cannot be efficiently simulated through classical computing. In our case with energy states, the objective is to find the solution that corresponds to the lowest energy, the global minimum of the shape of an energy function. Quantum tunneling is exploited in quantum annealing by passing through an high peak and thin energy hill, thus speeding up the search for the global minimum of the energy function, instead of climbing the energy hill itself [17].

Exploiting quantum tunneling for this solution search is like building a road tunnel into a mountain and pass through it instead of climbing all to way to the top and descend, it is a much faster, easier and "cheaper" (only from the energy spent point of view) process to obtain the same result. This behaviour is applied to solve NP-Hard problems and obtain acceptable solutions, but with the twist to leverage the quantum properties of qubits in the attempt to reduce the needed computational time.

As stated in the previous Section 2.6.1, the qubits' states and the final collapsed one depend on the qubits properties: first, how the qubits themselves are correlated between each other, then on their single magnetic field orientation. The correlation between qubits corresponds to the entanglement property, while the magnetic field orientation corresponds to the probability of each qubit's superposition. These measures are necessary to represent a qubit, and can be programmed and used to define the energy of the function to optimize.

How to represent quantum properties in a mathematical model?

As discussed multiple times, the objective to achieve with an annealer is to find the ground state, so the state with the minimal energy. This leads to the problem on how to calculate the energy of a quantum state.

The energy of the machine's state is obtained with its *Hamiltonian*, which is a mathematical description of the energy of a system obtained through the sum of the energies of its subparts, which in our case are the qubits and the interaction between themselves. The annealing process for quantum annealers is different with respect to the one of Simulated Annealing: in the simulated one, as we explained is Section 2.5.2, a temperature parameter is used to control the entire process and search for the global optimum. In Quantum Annealing, instead, the annealing process is not controlled with a temperature, but with two main parameters which are explained with the formula of the *Hamiltonian*. Here's the formula:

$$\mathcal{H} = \underbrace{-\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x^{(i)}\right)}_{\text{Initial Hamiltonian}} + \underbrace{\frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i>j} J_{ij}\hat{\sigma}_z^{(i)}\hat{\sigma}_z^{(j)}\right)}_{\text{Final Hamiltonian}} \tag{2.18}$$

where:

- s, the argument of $A(\cdot)$ and $B(\cdot)$, indicates the fraction of the annealing process, so it is a value between 0 and 1
- $\hat{\sigma}_{x,z}$ are the Pauli matrices:

$$\hat{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \hat{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \hat{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.19}$$

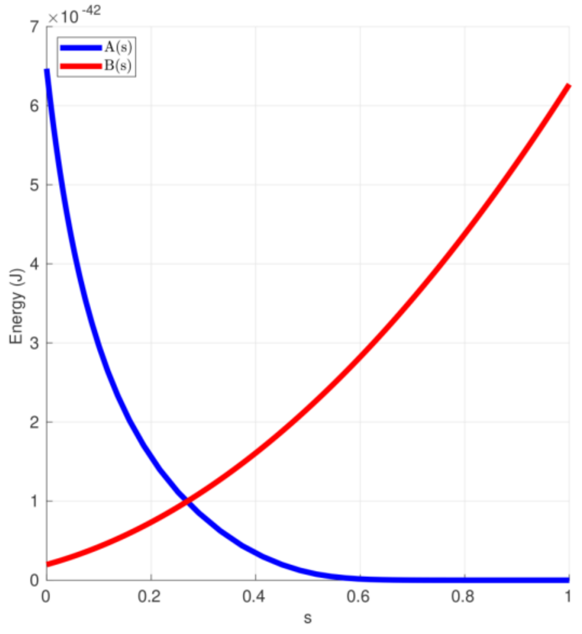


Figure 2.3: Evolution of functions A(s) and B(s) during the quantum annealing process.

- h_i is the *bias* of the i-th qubit, which represents its *superposition*
- J_{ij} is the *coupling strength* between the i-th and j-th qubit, which indicates the *entanglement* between them

$A(s)$ and $B(s)$ functions can be considered as "weights" of the sums in the parenthesis of the Hamiltonian formula (2.18), they control the annealing process and can be considered as analogous to the temperature schedule used in Simulated Annealing. These functions have different shapes, reported in Figure 2.3. As we can observe, these functions have the following behaviours: $A(s)$, which controls the *initial Hamiltonian* term, starts at a peak energy and it shrinks to zero as s tends to one, when the annealing schedule is getting to its end. $B(s)$ instead tends to be more and more present as the annealing schedule is finishing, indeed it describes the *final* or *problem Hamiltonian* term. This means that the lowest possible energy of the system, which coincides with the *ground state* (the solution that we are looking for), mainly depends on what happens at the end of the annealing schedule, so by the *problem* Hamiltonian.

Furthermore, the *problem* Hamiltonian is determined by the biases and the coupling strengths, which are set by the machine user. This means that if the machine user is able to build an optimization problem and map it as the *problem* Hamiltonian, the final result of the annealing schedule, the collapse of the qubits into a classical state, is a solution of the optimization problem.

In our discussion, we are still missing the link between the QUBO formulation problem, the one that we have stated that the Quantum Annealer is good at solving, and the Hamiltonian formula, which expresses the energy of a quantum system based on the biases and coupling strengths between the qubits. This missing link is the *Ising model*, which is the base model after which the problem Hamiltonian is defined.

The *Ising* model is a mathematical description of ferromagnetism in statistical mechanics, it consists of discrete variables that represent atomic "spin" property, that can assume the value of $+1$ or -1, which correspond to the spin directions. The energy function of the Ising model is the following, and it contains familiar variables:

$$E_{Ising}(s) = \sum_{i=1}^{N} h_i s_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} J_{ij} s_i s_j \tag{2.20}$$

As seen previously with the Hamiltonian (2.18), h_i indicates the bias of the i-th qubit q_i, J_{ij} the coupling strength between two qubits and N is the number of variables.

The Ising model can describe any Hamiltonian, and therefore also the problem Hamiltonian, so the energy of the final solution resulting from the annealing, where all the qubits have collapsed to a classical state. This means that if we are able to create an optimization problem that can be mapped to an Ising model, a solution can be obtained through annealing.

Furthermore, it becomes evident the bond with a QUBO formulation: the Ising formulation (2.20) coincides with a quadratic function that can be described with a $N \times N$ symmetric matrix, and the link with the Q matrix introduced in (2.13) is that the biases h_i form the diagonal of Q, while the coupling strengths J_{ij} correspond to the off-diagonal elements of the matrix.

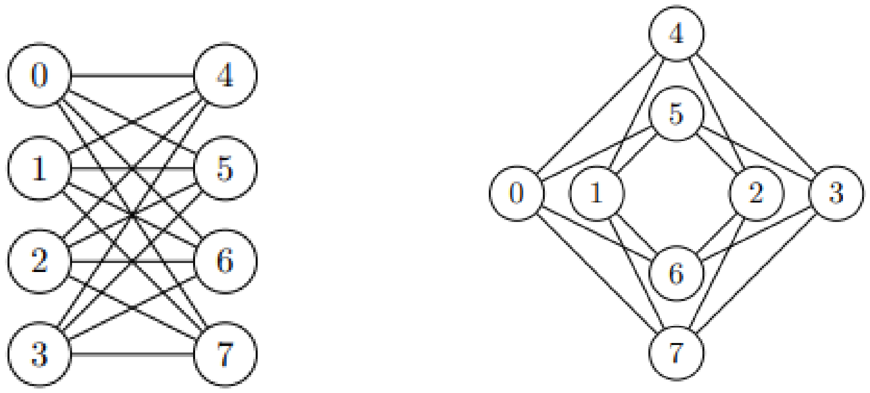
The only change that has to be performed regards the variables of the optimization problem, since QUBO variables are binary, while in Ising the variable is a spin, so +1 or -1. In fact, the spin s of the Ising model is related to the binary variable q of a QUBO problem as:

$$s_i = 2q_i - 1 \tag{2.21}$$

Problem embedding: from the mathematical problem to the Quantum annealer

Once the mathematical model is built, the next step is to "transfer" it onto the quantum annealer chip, in order to start the quantum annealing schedule and solve the optimization problem. From the square matrix to solving the problem with quantum annealing, there is an intermediate step called *problem embedding*. Before explaining it, an overview on the QPU architecture is needed. The QPU consists in an arrangement of qubits that are connected between each other. Several qubits are organised in *units*, arranged with a specific layout, called *topology*. A unit corresponds to a graph, the nodes of this graph are the qubits, the edges are the *couplers* and the arrangement of these edges is defined by the topology. A *coupler* is a link that connects two qubits, thus establishing an entanglement between them.

An example of a topology of quantum annealer machine unit is the *Chimera graph topology*, reported in Figure 2.4: it is composed by 8 nodes (qubits), which are not fully connected. This lack of couplers (edges) between some pair of qubits inside



(a) Bipartite graph (b) Cross graph

Figure 2.4: Chimera graph of a single unit of DW-2000Q-6

an unit raises an issue. In fact, going back to the mathematical model, the ideal scenario is to map each logical variable of the model to a single qubit, but as the dimensionality of the problem increases, this is no more possible. The square matrix that defines the quadratic objective function implies that there is a coupler (edge) for each pair of nodes, thus building a fully connected graph. As already seen, this is not the case with the topologies used.

This issue translates into the problem of fitting the structure of an Ising (or QUBO) formulation in the QPU topology. This process is called *minor embedding*

[15], which will trasform the original problem Hamiltonian into another that fits on the QPU, respecting the topology of its units. This process is itself a NP-Hard problem, but there are heuristic algorithms that are capable to obtain a solution in polynomial time, like the one described in [12].

Consider the following example: imagine a starting problem with 3 variables and an imaginary QPU unit of four qubits with a square topology. As it is shown in Figure 2.5, the QUBO formulation of the problem is described by a 3×3 symmetric matrix, generating a fully connected graph with three nodes (the logical variables) with a triangle structure. Due to the different structures, it is not possible to directly embed one problem variable to a single qubit. To solve this, one of the problem variables is duplicated and represented with two physical qubits. This step will generate a *chain* of qubits that represents the same problem variable. This embedding implies

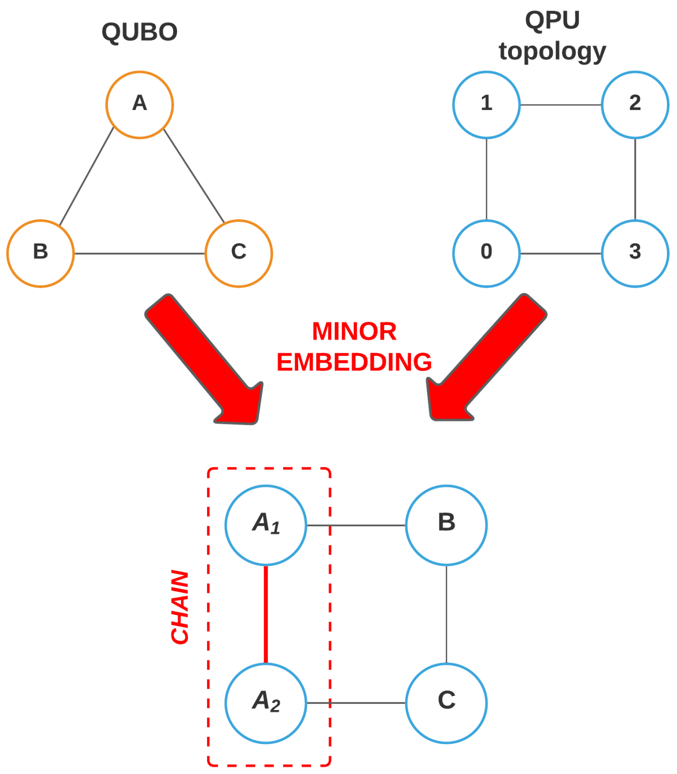


Figure 2.5: Minor embedding example with basic problem.

a change in the objective function, which will have an added penalty term associated to the generated chain. With minor embedding, in order to maintain consistency in the final results, all the qubits belonging to a chain must be equal.

Creating qubit chains to properly embed a QUBO problem onto the QPU has several consequences. With the previous example we have seen that, through embedding, multiple qubits are used to represent the same logical variable, thus the number of "free" variables that can be solved on the QPU descreases. Once the chains are created, an automated process adjusts the coupling strengths and biases. The coupling strengths of the qubits inside a chain is set to the maximum value available in order to maximize the entanglement between them.

When dealing with large problems, through minor embedding multiple and long qubit chains are created. These creation, along with the handling on all the parameters (coupling strengths and biases), increase the probability to obtain a *chain break*. A chain *breaks* if the qubits belonging to it have different values, this implies that the solution obtained is inconsistent. The final value of the logical variable can be decided via majority voting, but this leads to a solution that has a high probability of being not only suboptimal, but also to violate constraints.

The machine used: Quantum Annealer Advantage System

D-Wave Systems Inc.[1] is a Canadian quantum computing company, which sells their quantum computers' services to anyone, not only to corporates. The machine that has been used for the experiments in this thesis is the latest one available for cloud access through the company's portal Leap[2], the Advantage system[3], [4] that has been made available to the platform's users in september 2020. This quantum annealer has more than 5000 qubits, arranged on a QPU (Quantum Processing Unit) with a Pegasus topology [8].

With respect to the previous generation of D-Wave's quantum annealers, the Advantage system offers overall superior performances due to its higher connectivity, more qubits and couplers, thus providing the ability to embed larger problems. The topology of the single units is different, because with the previous generation (D-Wave 2000Q 6th gen., DW-2000Q-6) a Chimera topology is used. A graphical comparison, directly taken by D-Wave's documentation of the Advantage machine, of the two topologies is represented in Figure 2.6, where it is evident the increase of overall connectivity between the qubits. In these graphs, the edges between the qubits

[1]D-Wave Systems website: `https://www.dwavesys.com/`

[2]`https://cloud.dwavesys.com/leap/`

[3]Full Specifications on: `https://www.dwavesys.com/media/s3qbjp3s/14-1049a-a_the_d-wave_advantage_system_an_overview.pdf`

[4]Full documentation: `https://docs.dwavesys.com/docs/latest/index.html`

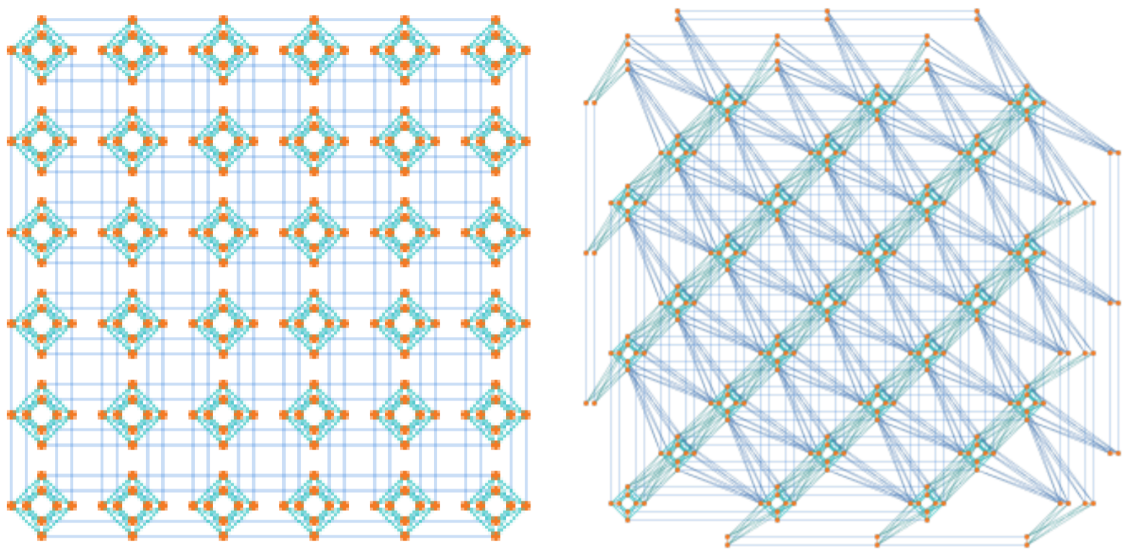


Figure 2.6: Chimera vs Pegasus topology

of the same unit or of different ones are couplers, thus their weights indicate the coupling strengths.

In this image there's only a partial representation of the complete architecture of the quantum machines; for example, the complete schema of the DW-2000Q-6 machine is reported in Figure 2.7, which presents a 16×16 matrix arrangement of Chimera units, each one consisting in a bipartite graph with 8 nodes that correspond to 8 qubits: its representation can be seen in Figure 2.4, which has been reported in the previous paragraph to introduce the QPU architecture. As stated previously, with Advantage quantum annealer it is possible to embed problems with a maximum of 5640 qubits: this translates for our case into a maximum fit for a dataset composed by around 120 features. For datasets above this value, D-Wave Systems provide a solution called *Hybrid Quantum-Classical Computing systems*.

Hybrid Quantum-Classical Computing systems

Solving large problems with many variables is an issue with these machines, since they are limited on the number of logical qubits embeddable into the QPU. That's where Hybrid Quantum-Classical systems can help.

With a problem of large dimensionality, for example a dataset with more than 120 features, this type of system decomposes the starting problem in subproblems in order to be solved efficiently, using both classical and quantum computing [49].

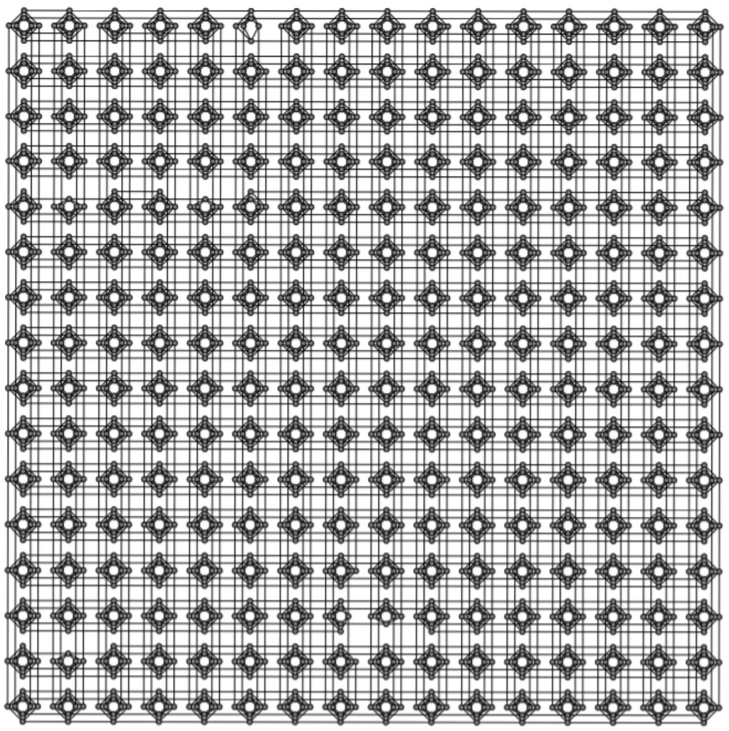


Figure 2.7: DW-2000Q-6 QPU topology

As for the previous approach with the Quantum annealer, the implementation and service is offered by Leap platform, D-Wave's cloud service.

Chapter 3

Quantum-based feature selection algorithms

The objective of this chapter is to describe in details the quantum-based feature selection algorithms that can be embedded onto a QPU of a Quantum Annealer.

As we've stated in the previous chapter, if we are able to define an optimization problem with an objective function that is quadratic, based on the symmetric Q matrix that defines it, we can formulate a QUBO problem that can be solved efficiently with a Quantum Annealer. The general schema of the algorithms described in this chapter until the embedding on the QPU [37] is the same:

1. define the formulas with statistical measures (e.g. mutual information, correlation and entropy) to build the matrix Q
2. create the matrix Q that describes the QUBO problem
3. embed the problem onto the QPU
4. solve the problem with the Quantum Annealer

Since we are dealing with the problem of feature selection, the final solution of the quantum annealing phase is an array of binary values, indicating which features are kept as relevant (1) or discarded (0).

These methods are going to be used in the fourth chapter "Experiments and Results" in order to evaluate their quality against each other and against classical methods.

3.1 Graph-based algorithm

The algorithm described is called *Graph Mutual Information QUBO* (*Graph-MIQUBO*), due to its main characteristics and output. It starts by building a fully connected weighted graph from the features of a dataset: the nodes are the features, and the weights of the edges are obtained by using the mutual information that has been introduced in Section 2.2.2. Then, this method outputs a QUBO formulation that can be then embedded and solved through quantum annealing.

The idea behind this method is to transform a feature selection problem into a weighted MaxCut one. The first step to obtain this comes by the article from Xuan Vinh Nguyen et al. [42], where a description of the MaxCut problem with a quadratic objective function is provided.

As addressed in the second chapter, a graph is described through the *adjacency* (or weight) matrix W which, in the case of a complete graph, consists in a square and symmetric matrix with zeros only on the main diagonal. As suggested by Z. Zhang and E. Hancock in 2011 [56], it is possible to determine a value between 0 and 1 indicating the *interaction* between two features, by using the mutual information (MI) from (2.5) and the entropy from (2.2). Thus, it is possible to build a square weight matrix W using the features of a dataset:

$$W = \begin{cases} W_{ij} = \frac{MI(f_i,f_j)}{H(f_i)+H(f_j)} & if \;\; i \neq j \\ W_{ij} = 0 & if \;\; i = j \end{cases} \tag{3.1}$$

where f_i is the i-th feature of the given dataset.

At this point we are able to define the QUBO problem. The quantum annealer solves a minimization problem and the weighted MaxCut is a maximization one, as explained in Section 2.4.1. To comply with this technicality it suffices to flip the sign of the objective function, thus obtaining the following formulation:

$$\min_{x^n} \quad -q(x) = -\left(x^T W x\right) \tag{3.2}$$

$$x \in \{0,1\}^n$$

The variable x is a vector containing n binary variables, where n is the number of features of the starting dataset. Notice that the ordering of the variables inside x and of the columns of W is determined by the original structure of the dataset itself. This implies that the first entry of vector x, so x_0, corresponds to the first column (or row) of matrix W and to the first feature f_0 of the dataset. The quadratic function

$q(x)$ is the one introduced in (2.13) and (2.14). Once the problem is written with this minimization formulation (3.2), it is ready to be given in input to the Quantum Annealer to embed it on its QPU architecture and then solve it.

3.2 Quantum-Boosting algorithm

The *Quantum-Boosting* (Q-Boosting) algorithm was first proposed in 2008 by Neven et al. [41] and then revisited and tested in 2009 and 2012 [40, 39].

As the name suggests, this algorithm exploits the Boosting technique [48, 22], which consists in the composition of very basic and weak learners trained on few features in order to obtain a strong learner that uses only relevant features.

For these reasons, the matrix Q that describes the QUBO problem to embed onto the QPU is the following:

$$\begin{cases} Q_{ij} = Corr(h_i, h_j) & if \;\; i \neq j \\ Q_{ij} = \frac{S}{N^2} + \lambda - 2 * Corr(h_i, y) & if \;\; i = j \end{cases} \tag{3.3}$$

where:

- $Corr(\cdot, \cdot)$ is the Pearson Correlation, defined in (2.1) in Section 2.2.1
- h_i coincides with the estimate of a Support Vector Machine (SVM) classifier trained **only** with the i-th feature of the analysed dataset
- y is the target variable
- S is the number of samples inside the dataset
- N is the number of features
- λ is an hyper-parameter of the algorithm, fixed a priori

3.3 Quantum-Correlation algorithm

Quantum-Correlation (Q-Correlation), introduced and tested by R. K. Nath et al. in 2008 [38], is another algorithm that creates a QUBO problem from the features of a dataset. More precisely, it computes the Pearson correlation (2.1) between the features and the target variable and between the features themselves.

In details, this is the formula to compute the matrix Q:

$$\begin{cases} Q_{ij} = Corr(f_i, f_j) & if \;\; i \neq j \\ Q_{ij} = Corr(f_i, y) & if \;\; i = j \end{cases} \tag{3.4}$$

where:

- $Corr(\cdot, \cdot)$ is the Pearson Correlation, defined in (2.1) in section 2.2.1
- f_i coincides with the i-th feature of the analysed dataset

Chapter 4

Experiments and Results

In this chapter we are going to present the experiments and the obtained results. As already mentioned in previous chapters, we are going to compare the three QUBO feature selection methods *Graph-MIQUBO*, *Q-Boosting* and *Q-Correlation* between themselves, as well as with the following classical filter methods, all explained in Section 2.2: *Variance Threshold*, *Mutual Information*, *Chi2 Test* and *ANOVA F-Test*.

For the three QUBO methods, the experiments are performed by solving the optimization problems with four solvers and approaches: *Simulated Annealing* (SA), *Tabu Search* (TS), *Steepest Descent* (SD) and *Quantum Annealing* (QA), with the usage of *Hybrid Quantum systems* instead of QA for datasets with over 124 features. The details about these listed approaches are reported in Sections 2.5.2 and 2.6.2 of the second chapter.

The results are going to be analysed from all perspectives, making comparisons and comments between the solvers on the same method, but also compare the solutions obtained by the different methods with respect to the same solver. This chapter is structured as follows:

- Section 4.1: the detailed pipeline of the workflow and an example of the execution is provided. This example shows specifically the execution of Graph-MIQUBO solved with both Simulated and Quantum Annealing over the iris dataset.

- Section 4.2: all the datasets used for the experiments are introduced and described.

- Section 4.3 and 4.4: the analysis and comparison of the classification accuracies obtained with the best classifiers across the solvers given a method, and across the methods (both quantum and non-quantum).

- Section 4.5: the confusion matrices computed with the best classifiers for some specific datasets are presented and commented.
- Section 4.6 and 4.7: comparison of the execution times needed to find the best classifiers for all QUBO methods across the solvers, and across all the algorithms (both quantum and non-quantum).

4.1 Workflow: experiments pipeline

In this section, the different steps of the workflow used for the experiments are presented.
To provide a quick overview, the pipeline starts with the splitting and preprocessing of the datasets into training and test sets used for the experiments. Then, the datasets are passed as input to start sequentially three feature selection methods that can be embedded onto the quantum annealer: our algorithm *Graph-MIQUBO*, *Q-Boosting* and *Q-Correlation*.

4.1.1 Dataset splitting and preprocessing

Each dataset used in the workflow is first handled to be split into training and test subsets, which are respectively used to train a classifier and to test its generalization capability with respect to unseen data. All the datasets used have been split with a 70% - 30% distribution of samples for training and testing. The validation set is generated from the training one, with the *k-fold cross-validation* method, explained in Section 4.1.2. The problems we are focusing on are supervised learning classification problems, so the splitting of a dataset into the subsets is performed by maintaining as close as possible the distribution percentages of the target classes. For example, if a dataset contains three classes distributed with this pattern:

- Class 0: 50%
- Class 1: 20%
- Class 2: 30%

Both training and testing sets will contain samples distributed with the a pattern as close as possible to the starting one.

4.1.2 Feature Selection methods applied steps

Overall, for all three methods implemented, the schema is the same. A block schema summarising all the steps is reported in Figure 4.1. All these steps for one dataset

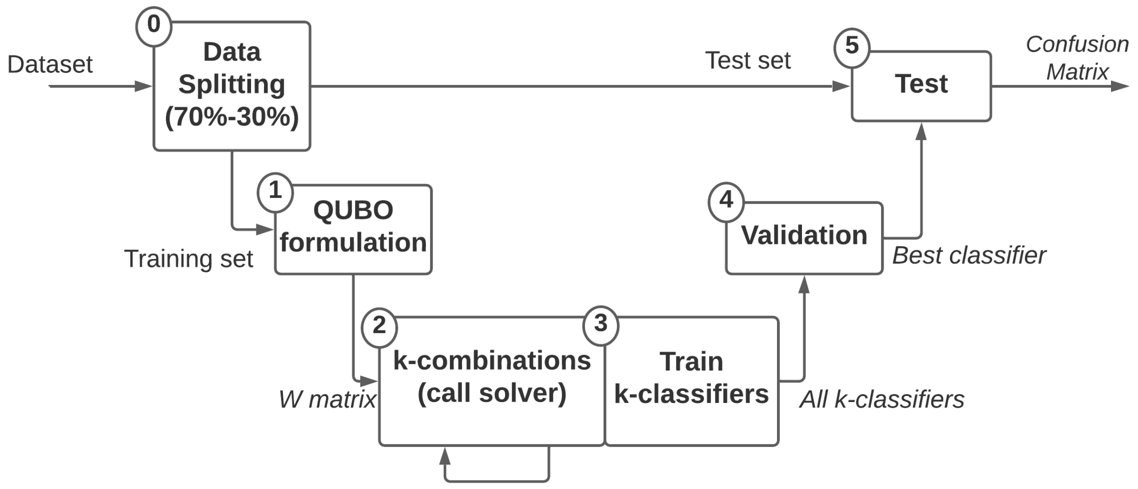


Figure 4.1: Block scheme of the evaluated methods' workflow

are performed with every feature selection method and for every solver used for the QUBO problem, thus the results will show each possible combination with Simulated Annealing, Tabu Search, Steepest Descent and Quantum Annealing (Hybrid in the case of datasets with over 124 features).

The details of these steps are addressed in the following paragraphs.

Step 1. QUBO formulation

After the data splitting phase into training and test set, the training set is given as input to the feature selection algorithms described and it first faces the computation of the square matrix.

In this thesis three methods are compared, each one with its logic to compute the quadratic objective function and respective square symmetric matrix, with the following references:

- *Graph-MIQUBO*: Equation (3.1), Section 3.1.
- *Q-Boosting*: Equation (3.3), Section 3.2.

- *Q-Correlation*: Equation (3.4), Section 3.3.

Before embedding the problem onto the QPU, the signs of the elements of the matrices are flipped, since each starting scenario describes a maximization problem: we need to formulate it as a minimization one in order to properly solve it with the quantum annealer, as explained in Section 2.6.2. This leads to flip the sign of the objective function, which suffices to convert a maximization formulation into a minimization one.

Step 2. K-combinations for QUBO

QUBO problems offer high flexibility, because by slightly changing the objective function it is possible to force the number of binary variables set to 1 in the solution. This is called *k-combinations*, and it is leveraged in our experiments. We need to perform this because some formulations of the optimization problem can be "unbalanced", which means that the final solution is a trivial one (e.g. all features or none of them are selected). This procedure adds to the general QUBO formulation (2.13) a penalty term in the objective function. This addition pushes the solver, no matter if classic like Simulated Annealing or Quantum, to select a certain percentage of the variables, thus having more corresponding 1 (ones) in the solution found.

While using k-combinations for QUBO problems, the modified objective function is the following:

$$min \quad q(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i \, x_j \, Q_{ij} + strength * \left(\sum_{i=0}^{n-1} x_i - k_c \right)^2 \tag{4.1}$$

$$x \in \{0;1\} \quad \forall i,j = 0,1,...,n-1$$

The variable *strength* is an hyper-parameter fixed a priori that controls the magnitude of the imposed penalty. The parameter k_c is a positive integer ranging from 1 to n, the dimensionality of the problem (e.g. the features of the original dataset). From the solution point of view, adding these penalties will result in the solver finding a solution with **exactly** k_c variables selected, so set to 1.

In our algorithm, since datasets of different sizes and number of features are used, the total number of k-combinations problems formulated for each dataset is set up to maximum of 50 different combinations, to prevent an extensive usage of computing resources. If the number of features n of a dataset is higher than 50, then the 50 values of k_c are equivalently spaced between 1 and $n-1$. Whereas, if n is lower than 50, the integer values between 1 and $n-1$ are used as k_c. For example: given $n = 256$, the values of k_c are:

$$[1, 6, 11, 16, 21, \ldots 229, 234, 239, 244, 249, 255]$$

Each solution of a k-combination problem computed by the solver, consisting in a binary vector of 0s and 1s, will be passed as input to a classifier used to *validate* the generalization power of the model with the features selected.

Step 3. Classifier

Once we have all the solutions computed with the solver, each one is used to filter the training set of the dataset accordingly and used to train a classifier.

In this thesis, the classifier used is the Random Forest, which has been explained in Section 2.3.1. For each k_c*-th* solution computed at the previous step, a filtered dataset is generated that is going to be used to train a Random Forest classifier. This implies that one Random Forest is instantiated for each solution computed. Thus, the outcome of this step is a set of Random Forests, each corresponding to a filtered and reduced dataset.

Step 4. Validation: find the best classifier

Given the collection of all the classifiers each with the respective reduced dataset, we need to determine the most promising from a generalization point of view. To achieve this, we need to assign a score to each classifier in order to determine the best one.

To assign a score, the criteria used is based on the *stratified k-fold cross-validation* method: the training set is equally divided in k_f parts, maintaining also the distribution of the target classes as close as possible in each fold. During its iterations, $k_f - 1$ folds are used as training subsets while one is used as a validation subset, over which the accuracy of the classifier is computed. This process is repeated k_f times, meaning that all the folds are used once as validation subsets.

In our experiments, the number k_f of folds is set to 5 and the score given to each instantiated classifier is the mean of the classification accuracies over the validation folds, consisting of a real value between 0 and 1. The winning classifier is the one that has the highest classification accuracy, which we will refer to as *cross-validation score*.

Step 5. Testing

The final step of this workflow is the *testing* phase of the best classifier obtained as output of the previous step. We need to highlight the fact that the test set obtained by the splitting of the original dataset has never been used or ”seen” by the classifier during its training phase. The test set is in fact used in this final step to prove the generalization capability of the best Random Forest found in the validation step by computing the *confusion matrix*.

Given as C the set of target classes to learn, the confusion matrix is a square $|C| \times |C|$ matrix that presents on the diagonal the number of samples correctly classified, while the number of misclassified samples per class are reported as off-diagonal elements.

The closest to a diagonal matrix the confusion matrix is, the higher is the accuracy of the best learner found over unseen data during the training phase and so its generalization power to correctly classify new incoming samples.

4.1.3 Example: Graph-MIQUBO solution with Simulated and Quantum Annealing

The iris dataset, downloaded with the sklearn [43] Python package[1] and showed in Table 4.1, cointains 150 samples and 4 features, which are all numerical and of float type:

- sepal length (cm)
- sepal width (cm)
- petal length (cm)
- petal width (cm)

This is a quick overview of the dataset itself, only with the features. The target variable consists of three labels (0, 1, 2) indicating respectively three species of iris flowers: Iris setosa, Iris virginica and Iris versicolor.

Having 4 features means that the W adjacency weight matrix for the complete graph

[1]`https://scikit-learn.org/stable/index.html`

#Sample	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	...	...	...	...
⋮	⋮	⋮	⋮	⋮
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Table 4.1: Iris dataset

for the iris dataset, computed with the formula (3.1), is a 4×4 symmetric matrix:

$$W = \begin{bmatrix} 0.0 & 0.23627 & 0.30466 & 0.25256 \\ 0.23627 & 0.0 & 0.24591 & 0.2076 \\ 0.30466 & 0.24591 & 0.0 & 0.29665 \\ 0.25256 & 0.2076 & 0.29665 & 0.0 \end{bmatrix} \tag{4.2}$$

By using the weight matrix W computed above (4.2), we can formulate the QUBO problem, and find a solution with one of the solvers listed and described in previous chapters. Thus, the algorithm passes to the step where the different solvers are used to find the solution coinciding with the ground state, which is the one with the lowest energy.

In Tables 4.2 and 4.3 the solutions found with Quantum Annealing (QA) and Simulated Annealing (SA) approaches with the corresponding energies are reported. In these tables, the first three columns indicate respectively the variables (so, the features) set to 0, the variables set to 1, and the corresponding state. The annealing process, for both solvers, is repeated 1000 times each reporting a solution with the minimum energy found. The final solution chosen is the state with the lowest energy. The number of times that the same solution is found is reported under the column labeled as "Occurrences". The final solutions found by both QA and SA can be also visualized in the form of histograms in Figures 4.2 and 4.3. In these histograms, on the x-axis are reported the energies of the solution states found, and on the y-axis the occurrences.

As we can observe from the referenced tables with the solutions, the state with the lowest energy, which is marked with a yellow row, is the same for both solvers. This means that we are able to find the same conclusion: the second feature "sepal

Quantum Annealing (QA)				
$x_i = 0$	$x_i = 1$	State	Energy	Occurrences
f1	f0, f2, f3	**[1 0 1 1]**	-0.83888	658
f3	f0, f1, f2	[1 1 1 0]	-0.74214	177
f0	f1, f2, f3	[0 1 1 1]	-0.67196	45
f1, f3	f0, f2	[1 0 1 0]	-0.64795	28
f0, f1	f2, f3	[0 0 1 1]	-0.63977	25
f0, f3	f1, f2	[0 1 1 0]	-0.56054	10
f2	f0, f1, f3	[1 1 0 1]	-0.55646	24
f1, f2	f0, f3	[1 0 0 1]	-0.55116	12
f2, f3	f0, f1	[1 1 0 0]	-0.53365	18
f0, f2	f1, f3	[0 1 0 1]	-0.47165	3

Table 4.2: Graph-MIQUBO applied to iris dataset: Quantum Annealing solutions found with energy and number of occurrences. The first three columns indicate respectively the variables (so, the features) set to 0, the variables set to 1, and the corresponding state. The first yellow row indicates the final solution, so the state found with the minimum energy.

Simulated Annealing (SA)				
$x_i = 0$	$x_i = 1$	State	Energy	Occurrences
f1	f0, f2, f3	**[1 0 1 1]**	-9.85387	472
f3	f0, f1, f2	[1 1 1 0]	-9.78684	244
f0	f1, f2, f3	[0 1 1 1]	-9.75016	162
f2	f0, f1, f3	[1 1 0 1]	-9.69643	105
-	f0, f1, f2, f3	[1 1 1 1]	-9.54365	17

Table 4.3: Graph-MIQUBO applied to iris dataset: Simulated Annealing solutions found with energy and number of occurrences. The first three columns indicate respectively the variables (so, the features) set to 0, the variables set to 1, and the corresponding state. The first yellow row indicates the final solution, so the state found with the minimum energy.

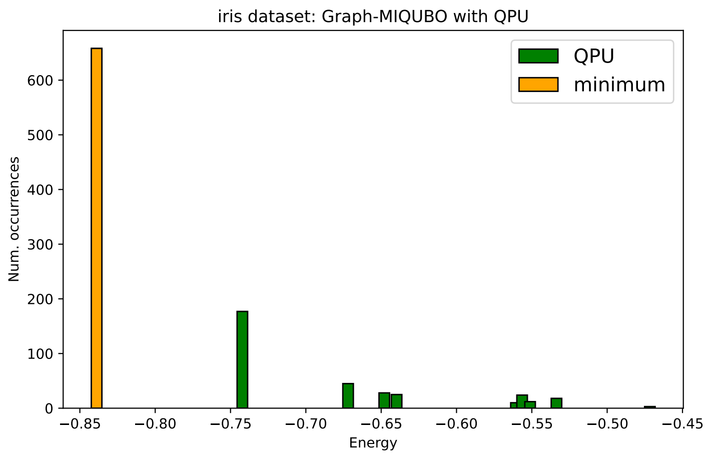


Figure 4.2: Graph-MIQUBO applied to iris dataset: QA solutions found with histogram energy and number of occurrences. In orange the solution with the minimum energy.

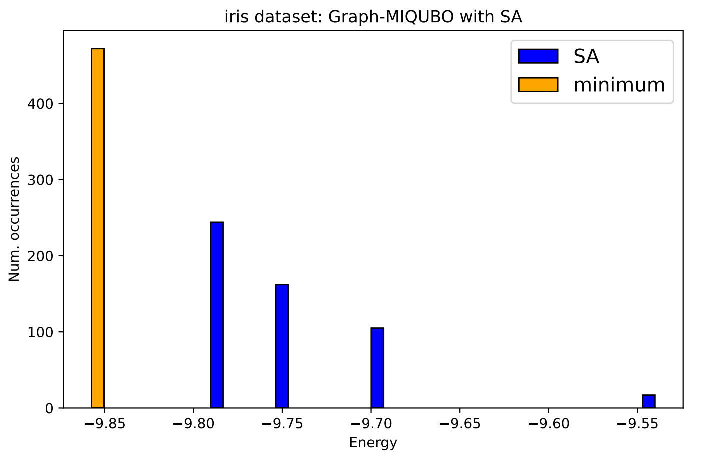


Figure 4.3: Graph-MIQUBO applied to iris dataset: SA solutions found with histogram energy and number of occurrences. In orange the solution with the minimum energy.

width" of the iris dataset is the least relevant in order to learn and predict the target class.

Furthermore, we can show the histograms of how many features during all k-combinations were selected (so with the corresponding x_i set to 1) and, finally, which ones are discarded or not with the computation of the best scoring classifier in the validation phase. This representation is provided in Figure 4.4, where on the y-axis is reported the frequency of features selected in the solutions found by the solver (for the referenced figure, the approach used is the Quantum Annealing), and on the x-axis the features of the dataset: in green the bars corresponding to the features selected of the best scoring classifier, in red the features discarded. In the case of the iris dataset, using Graph-MIQUBO and Quantum Annealing, out of four total features only one is discarded and is "sepal width": as we can see by this frequency histogram, the discarded feature is never been assigned a "1" in one of the solutions of k-combinations. We can visualize also how the Random Forest classifier trained

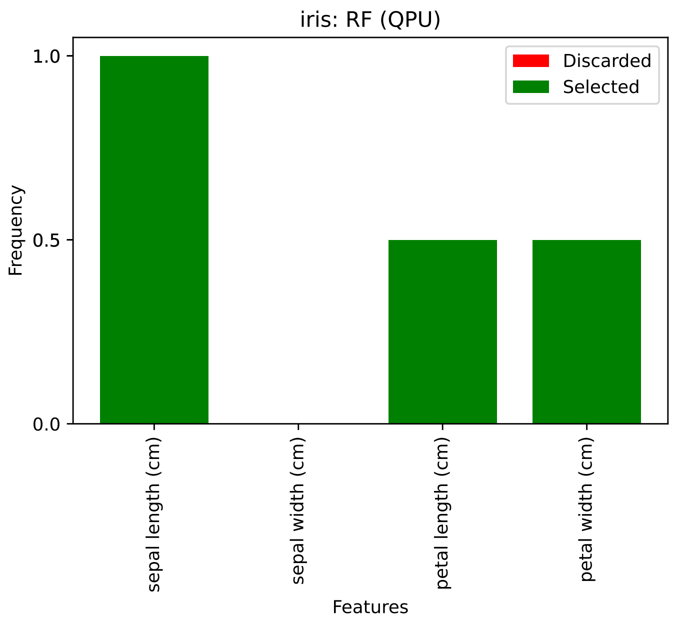


Figure 4.4: Iris dataset: histogram of solutions of Graph-MIQUBO solved with Quantum Annealing (QA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded. In this case, the feature discarded "sepal width", is never chosen.

with this reduced dataset performs during the test phase with the confusion matrix. The visualization of the confusion matrix is reported in Figure 4.5, and it is obtained with only the features selected with the best classifier using the unseen test set.

From this confusion matrix, it is possible to see that by using the test set the trained Random Forest misclassifies only one sample out of 45.

The description of all the datasets used in this thesis is presented in the following

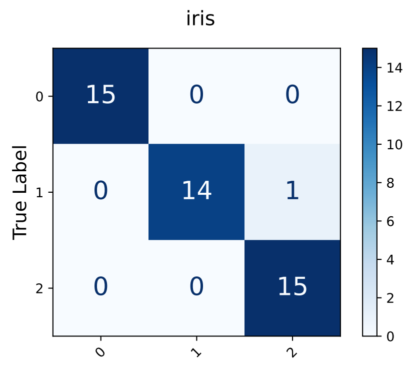


Figure 4.5: Confusion Matrix on test set of iris dataset with features selected by Graph-MIQUBO with the QPU.

section, after which all the results obtained with them will be presented.

4.2 Datasets used

In this section an overview of all the datasets used for the experiments is reported. Specifically, 18 datasets have been used, which have been downloaded from Openml.org[2], a website where open datasets of all types and dimensionalities are available to users, free with no subscription.

In Table 4.4, all the properties of the 18 datasets are reported: the number of total features, the number of samples and how many of them are divided between the training set (70%) and testing set (30%), as well as the number of different classes that represent the target variable for each one of the dataset. The last column, labeled as "*Balance*", corresponds with a tag regarding the balance of presence of the classes in the target variable array. For instance, the iris dataset is tagged as "balanced" since it has 3 classes and they are all equivalently split in the target array, whereas

[2]`https://www.openml.org/`

the thyroid-ann is ”very unbalanced”, since the percentage of samples belonging to the same class is far greater with respect to the other classes’ (92,5% against 2,5% and 5%). The details with the percentages of the classes for each dataset is reported in Table 4.5. The datasets chosen for the experiments cover a variety of domains:

		Samples			**Classes**	
Dataset	Features	Total	Training [70%]	Test [30%]	#	Balance
iris	4	150	105	45	3	balanced
breast_cancer	30	569	398	171	2	unbalanced
wine	13	178	124	54	3	balanced
vehicle	18	846	592	254	4	balanced
ionosphere	34	351	245	106	2	balanced
robot-failures-lp5	90	164	114	50	5	unbalanced
waveform-5000	40	5000	3500	1500	3	balanced
steel-plates-fault	33	1941	1358	583	2	unbalanced
nomao	118	34465	24125	10340	2	unbalanced
SPECTF	44	267	186	81	2	unbalanced
cars1	7	392	274	118	3	unbalanced
LED-7digit	7	500	350	150	10	balanced
thyroid-ann	21	3772	2640	1132	3	very unbalanced
spambase	57	4601	3220	1381	2	unbalanced
tecator	124	240	168	72	2	balanced
isolet	617	600	420	180	2	balanced
USPS	256	1424	996	428	2	balanced
swarm-behaviour	2400	24016	16811	7205	2	unbalanced

Table 4.4: List of all the datasets used for the experiments with main properties. There are reported the total number of features and samples, as well as the cardinality of the training and testing sets. In the two columns on the right, There is reported the number of different classes in the target variable and a ”Balance” tag, that indicates the balance of presence of the classes in the target variable array. For instance, for ”thyroid-ann” dataset the tag is ”very unbalanced” because out of three classes, one corresponds to more than 90% of the samples.

from flowers like ”iris”, animals like ”swarm-behaviour” and ”tecator”, to industrial processes as ”robot-failures-lp5”, ”steel-plates-fault”, as well as medical ones with ”breast_cancer”, ”SPECTF” (cardiovascular diagnosis) and ”thyroid-ann”. All these datasets cover a variety of different scenarios: datasets with different amount of features, with low or very high number of samples and also with different balances of classes percentages in the target variable.

Dataset	#Classes	Tag	Balance
iris	3	balanced	33.33% 33.33% 33.33%
breast_cancer	2	unbalanced	37.19% 62.81%
wine	3	balanced	33.06% 40.32% 26.61%
vehicle	4	balanced	25.84% 25.0% 25.68% 23.48%
ionosphere	2	balanced	35.92% 64.08%
robot-failures-lp5	5	unbalanced	15.79% 13.16% 28.95% 15.79% 26.32%
waveform-5000	3	balanced	33.83% 33.06% 33.11%
steel-plates-fault	2	unbalanced	65.32% 34.68%
nomao	2	unbalanced	28.56% 71.44%
SPECTF	2	unbalanced	20.43% 79.57%
cars1	3	unbalanced	17.52% 20.07% 62.41%
LED-7digit	10	balanced	$\approx 10.0\%(mean)$
thyroid-ann	3	very unbalanced	2.46% 5.08% 92.46%
spambase	2	unbalanced	60.59% 39.41%
tecator	2	balanced	57.74% 42.26%
isolet	2	balanced	50.0% 50.0%
USPS	2	balanced	50.3% 49.7%
swarm-behaviour	2	unbalanced	62.5% 37.5%

Table 4.5: List of all the datasets used for the experiments with details of the balance of the target classes occurrences.

4.3 Comparison across the QUBO solvers

In this section, the results obtained with the listed datasets are presented in order to compare the best solutions obtained with the different solvers given a QUBO feature selection method. Thus, we present three tables, one for each method containing the four solvers used.

Solvers:

- *Quantum Annealing* (QA): for the corresponding column, it is reported in parenthesis the values of *chain break fractions*. This is a float value between 0 and 1 indicating the proportion of qubit chains, created during the embedding, that broke during the search for the state with minimum energy. For the last three datasets, which all have more than 124 features, the Hybrid quantum system has been used instead. With the hybrid approach we are not able to determine the number (or fraction) of chains that are broken during the execution.

- *Simulated Annealing* (SA)
- *Tabu Search* (TS)
- *Steepest Descent* (SD)

The first table reported (Table 4.6, page 44) shows the results obtained with *Graph-MIQUBO*. As we can see, across the different solvers, the performance is consistent: for most of the datasets the overall cross-validation score (labeled as "CV_score") obtained with the features selected by the algorithm is equal or higher with respect to the accuracy with all the features of the original dataset. Furthermore, it is noticeable that also the number of features discarded is consistent across the solvers, showing that the solutions obtained with Quantum Annealing are in line with the ones computed with non-quantum solvers. For example, a promising result is obtained for the medical datasets, where the algorithm is able to nearly half of the features for "breast cancer" and "thyroid", and to discard 35 out of 44 features for the "SPECTF" dataset, while obtaining improved accuracy and results similar if not better with respect to the non-quantum solvers.

The last three rows coincide with the datasets where the hybrid approach has to be adopted: here we see that the performances are consistent for the "USPS" and "swarm-behaviour" dataset, obtaining an higher or equal accuracy score no matter the solver. Instead, for the "isolet" one, the QPU struggles to find a bigger set of features to discard, where with Simulated Annealing nearly 99% of features are cut, but still maintaining the same accuracy.

Graph-MIQUBO										
	All Features		QA		SA		TS		SD	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924 (0.0)	3	0.933	3	0.924	3	0.924
breast_cancer	30	0.957	**17**	**0.957 (0.0)**	19	0.962	15	0.957	29	0.960
wine	13	0.984	12	0.992 (0.0)	7	0.976	10	0.976	10	0.976
vehicle	18	0.753	16	0.753 (0.0)	17	0.747	16	0.748	14	0.748
ionosphere	34	0.918	32	0.931 (0.0)	32	0.935	32	0.935	33	0.935
robot-failures-lp5	90	0.702	70	0.754 (0.48)	74	0.746	80	0.746	63	0.746
waveform-5000	40	0.851	38	0.853 (0.07)	24	0.853	14	0.852	19	0.854
steel-plates-fault	33	0.989	32	0.982 (0.0)	32	0.983	32	0.982	32	0.982
nomao	118	0.966	**92**	**0.966 (0.57)**	106	0.967	105	0.966	113	0.967
SPECTF	44	0.790	**9**	**0.817 (0.23)**	9	0.817	33	0.822	17	0.828
cars1	7	0.799	6	0.803 (0.0)	5	0.814	5	0.811	5	0.811
LED-7digit	7	0.689	**6**	**0.717 (0.0)**	6	0.714	6	0.706	6	0.706
thyroid-ann	21	0.995	**12**	**0.997 (0.0)**	18	0.997	18	0.997	19	0.997
spambase	57	0.948	55	0.949 (0.05)	53	0.949	55	0.950	56	0.949
tecator	124	0.917	**103**	**0.929 (0.57)**	123	0.917	123	0.923	123	0.881
USPS	256	0.984	241	0.988	224	0.989	237	0.990	240	0.990
isolet	617	0.979	539	0.986	5	0.986	365	0.986	473	0.986
swarm-behaviour	2400	1.000	**746**	**1.0**	746	1.000	746	1.000	746	1.000

Table 4.6: Graph-MIQUBO results across the solvers. In the *cross-validation score* (CV_score) field of the QPU column, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach. Approaches used to solve QUBO problems: Quantum Annealing (QA), Simulated Annealing (SA), Tabu Search (TS) and Steepest Descent (SD). Highlighted in bold the results where QA provided the highest number of features discarded, and better (or equal) score with respect to the classifiers trained with all features and trained using the solutions of the other solvers.

A different scenario is obtained with *Q-Boosting*, which results are reported in Table 4.7 (page 46). In this table, we can observe that the number of cut features is consistent across the solvers, even with the hybrid quantum one with only some struggling with the "isolet" dataset. In general, at least 20%-25% of features are discarded by all different solvers. This results are not correlated with the type of the analysed dataset: if we focus on the last three ones with hundreds of features, the number of discarded features is not related to the balance of classes percentages in the target variable. For instance, "USPS" and "isolet" are both balanced, but on the former only 18 out of 256 features are discarded, whereas with the latter Q-Boosting is able to discard from 35% (only with Tabu Search) to 94% of the features maintaining equal accuracies.

The final table, the one that shows the results of *Q-Correlation* (page 47), presents similar results with respect to the Q-Boosting one: the performances across the different solvers are comparable, even if for most datasets the algorithm struggles to find a subset of features that allows to obtain an accuracy of the classifier in the validation phase that is equal or higher to the one computed by using all the features. Furthermore, it seems that it is not able to discard more than 15% to 20% of features. This does not apply for the results obtained for the last three large datasets: with the exception of "USPS" dataset where Q-Correlation discards only 10-11 features (4% of the total), for "isolet" and "swarm-behaviour" from 72% to 89% of the features are cut and still the classification accuracies are equal or higher.

Q-Boosting										
	All Features		QA		SA		TS		SD	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924 (0.0)	3	0.924	3	0.924	3	0.924
breast_cancer	30	0.957	25	0.96 (0.0)	25	0.960	22	0.960	19	0.957
wine	13	0.984	11	0.976 (0.0)	12	0.968	11	0.968	11	0.976
vehicle	18	0.753	16	0.74 (0.0)	17	0.748	17	0.748	17	0.745
ionosphere	34	0.918	25	0.939 (0.06)	19	0.939	18	0.943	18	0.939
robot-failures-lp5	90	0.702	80	0.745 (0.31)	66	0.746	77	0.755	61	0.737
waveform-5000	40	0.851	**32**	**0.854 (0.35)**	38	0.846	37	0.845	37	0.845
steel-plates-fault	33	0.989	28	0.992 (0.09)	32	0.992	10	0.998	15	0.996
nomao	118	0.966	**83**	**0.967 (0.65)**	103	0.967	116	0.967	91	0.967
SPECTF	44	0.790	12	0.838 (0.32)	6	0.844	5	0.844	17	0.833
cars1	7	0.799	**4**	**0.814 (0.0)**	5	0.810	4	0.807	4	0.807
LED-7digit	7	0.689	**5**	**0.709 (0.0)**	5	0.709	5	0.709	5	0.706
thyroid-ann	21	0.995	**11**	**0.997 (0.0)**	16	0.997	13	0.997	14	0.996
spambase	57	0.948	**40**	**0.949 (0.46)**	55	0.948	56	0.951	48	0.948
tecator	124	0.917	120	0.905 (0.03)	120	0.905	120	0.911	120	0.917
USPS	256	0.984	238	0.987	239	0.990	222	0.990	230	0.988
isolet	617	0.979	114	0.986	57	0.986	397	0.988	38	0.986
swarm-behaviour	2400	1.000	**171**	**1.0**	171	1.000	171	1.000	171	1.000

Table 4.7: Q-Boosting results across the solvers. In the *cross-validation score* (CV_score) field of the QPU column, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach. Approaches used to solve QUBO problems: Quantum Annealing (QA), Simulated Annealing (SA), Tabu Search (TS) and Steepest Descent (SD). Highlighted in bold the results where QA provided the highest number of features discarded, and better (or equal) score with respect to the classifiers trained with all features and trained using the solutions of the other solvers.

Q-Correlation										
	All Features		QA		SA		TS		SD	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.933 (0.0)	3	0.933	3	0.933	3	0.933
breast_cancer	30	0.957	28	0.96 (0.0)	29	0.960	27	0.957	27	0.957
wine	13	0.984	10	0.984 (0.0)	8	0.976	10	0.984	8	0.968
vehicle	18	0.753	15	0.748 (0.0)	16	0.747	13	0.745	15	0.752
ionosphere	34	0.918	30	0.931 (0.03)	23	0.931	25	0.927	30	0.939
robot-failures-lp5	90	0.702	**19**	**0.754 (0.39)**	77	0.745	74	0.771	76	0.737
waveform-5000	40	0.851	35	0.818 (0.05)	33	0.813	35	0.816	35	0.820
steel-plates-fault	33	0.989	**29**	**0.993 (0.0)**	29	0.992	29	0.991	29	0.990
nomao	118	0.966	**82**	**0.967 (0.69)**	85	0.967	90	0.967	99	0.967
SPECTF	44	0.790	36	0.817 (0.02)	36	0.817	14	0.806	14	0.806
cars1	7	0.799	5	0.821 (0.0)	5	0.814	5	0.825	4	0.818
LED-7digit	7	0.689	5	0.603 (0.0)	5	0.611	5	0.614	5	0.609
thyroid-ann	21	0.995	5	0.946 (0.0)	5	0.947	19	0.947	19	0.947
spambase	57	0.948	51	0.948 (0.16)	48	0.945	49	0.946	53	0.945
tecator	124	0.917	117	0.881 (0.02)	117	0.893	116	0.893	116	0.899
USPS	256	0.984	245	0.983	245	0.985	246	0.986	245	0.986
isolet	617	0.979	174	0.986	64	0.986	70	0.986	33	0.986
swarm-behaviour	2400	1.000	**257**	**1.0**	257	1.000	257	1.000	257	1.000

Table 4.8: Q-Correlation results across the solvers. In the *cross-validation score* (CV_score) field of the QPU column, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach. Approaches used to solve QUBO problems: Quantum Annealing (QA), Simulated Annealing (SA), Tabu Search (TS) and Steepest Descent (SD). Highlighted in bold the results where QA provided the highest number of features discarded, and better (or equal) score with respect to the classifiers trained with all features and trained using the solutions of the other solvers.

In all three tables mentioned, in the columns related to the results computed with Quantum Annealing, the chain break fraction tends to rise with a value between 0.60 and 0.70, corresponding to 60% and 70%, for the datasets in which the number of features gets closer to 124. This behaviour is expected, since we know that the size of a problem that can be embedded onto the QPU cannot be infinite. This means that the results obtained with quantum annealing can be considered reliable only if the corresponding value of chain break fraction is low enough.

4.4 Comparison across the algorithms

In this section we are going to visualize the same results shown in Section 4.3, but with a different perspective: we will analyse a table containing the information of the cross-validation score when all features are used, and the best solutions found by the three QUBO algorithms solved with Quantum Annealing. Thus, we can see the performance between the different methods directly.

Furthermore, in a separate subsection we report the tables showing a comparison of the results obtained with Quantum Annealing for the three QUBO feature selection methods against the classical methods mentioned at the beginning of this chapter.

4.4.1 QUBO algorithms

Here is provided the table of the results obtained with the three QUBO methods solved with Quantum Annealing. We are only focusing the results obtained with the QPU, all the tables with the results obtained with the other solvers are reported in Section A.1 of the Appendix.

Following this explanation, we focus on Table 4.9: overall, we can observe that with all three QUBO algorithms is obtained an improvement of the classification task because all the accuracies of the best classifiers found are equal or higher with respect to the one trained using all the features. Additionally, with Graph-MIQUBO we are able to embed efficiently problems and maintain in general the highest accuracy score across the methods, but discards less features with respect to the other QUBO feature selection methods. Instead, with Q-Boosting and Q-Correlation, more features are discarded, thus with lighter and more compact filtered datasets, but with the cost of obtaining less overall classification accuracy in validation. This behaviour can be seen with ”wine”, ”vehicle”, ”LED-7Digit” and ”thyroid” datasets. It is important

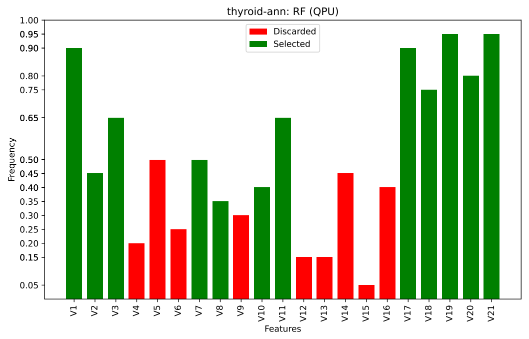
Quantum Annealing (QA)								
	All Features		Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924 (0.0)	3	0.924 (0.0)	3	0.933 (0.0)
breast_cancer	30	0.957	**17**	**0.957 (0.0)**	25	0.96 (0.0)	28	0.96 (0.0)
wine	13	0.984	12	0.992 (0.0)	11	0.976 (0.0)	10	0.984 (0.0)
vehicle	18	0.753	16	0.753 (0.0)	16	0.74 (0.0)	15	0.748 (0.0)
ionosphere	34	0.918	32	0.931 (0.0)	**25**	**0.939 (0.06)**	30	0.931 (0.03)
robot-failures-lp5	90	0.702	70	0.754 (0.48)	80	0.745 (0.31)	**19**	**0.754 (0.39)**
waveform-5000	40	0.851	38	0.853 (0.07)	32	0.854 (0.35)	35	0.818 (0.05)
steel-plates-fault	33	0.989	32	0.982 (0.0)	28	0.992 (0.09)	29	0.993 (0.0)
nomao	118	0.966	92	0.966 (0.57)	83	0.967 (0.65)	**82**	**0.967 (0.69)**
SPECTF	44	0.790	**9**	**0.817 (0.23)**	12	0.838 (0.32)	36	0.817 (0.02)
cars1	7	0.799	6	0.803 (0.0)	4	0.814 (0.0)	5	0.821 (0.0)
LED-7digit	7	0.689	6	0.717 (0.0)	5	0.709 (0.0)	5	0.603 (0.0)
thyroid-ann	21	0.995	12	0.997 (0.0)	11	0.997 (0.0)	5	0.946 (0.0)
spambase	57	0.948	55	0.949 (0.05)	**40**	**0.949 (0.46)**	51	0.948 (0.16)
tecator	124	0.917	**103**	**0.929 (0.57)**	120	0.905 (0.03)	117	0.881 (0.02)
USPS	256	0.984	241	0.988	238	0.987	245	0.983
isolet	617	0.979	539	0.986	**114**	**0.986**	174	0.986
swarm-behaviour	2400	1.000	746	1.0	**171**	**1.0**	257	1.0

Table 4.9: Quantum Annealing approach results across the algorithms. In the *cross-validation score* (CV_score) field of the method columns, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach. Highlighted in bold the results where the QUBO feature selection method provided the highest number of features discarded, and better (or equal) score with respect to the classifiers trained with all features and trained using the solutions of the other algorithms.

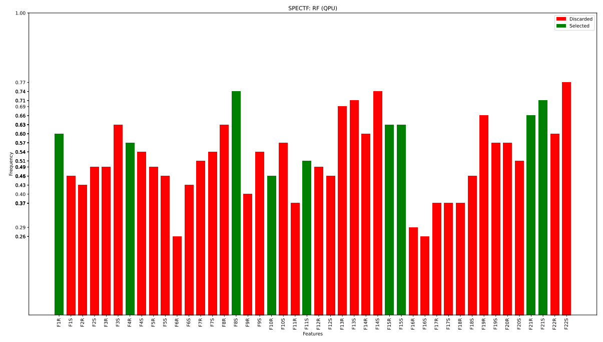
to state that we cannot infer a general rule of the best solution obtained based on the solutions computed of k-combinations: the behaviours of the three methods solved with Quantum Annealing are not perfectly distinct. We can provide an example by showing the solutions obtained on the datasets "thyroid" and "SPECTF". This example is shown in three figures, one for each QUBO method:

1. Graph-MIQUBO: referred to as *G-MIQUBO*, Figure 4.6
2. Q-Boosting: referred to as *Q-Boost*, Figure 4.7
3. Q-Correlation: referred to as *Q-Corr*, Figure 4.8

Each image shows on the left the "thyroid" dataset, while on the right the "SPECTF" one. While we can observe that for the "thyroid" dataset all three methods tend to cut features below a certain frequency and always picking the ones with the highest bars, with only few exceptions, for ”SPECTF” we can see that the behaviour is overall different. In fact, for the this dataset, except fot the Q-Correlation image (Figure 4.8b), the features discarded are also the ones that are particularly frequent during the computation of k-combinations.



(a) Thyroid: G-MIQUBO (QA).



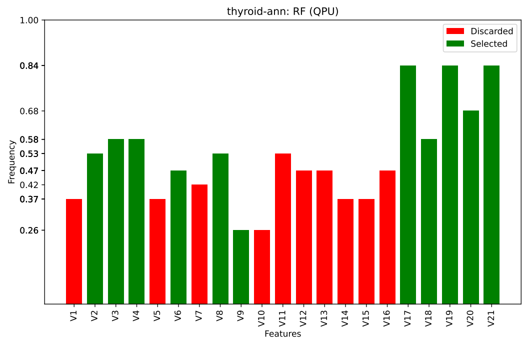
(b) SPECTF: G-MIQUBO (QA).

Figure 4.6: Thyroid vs SPECTF dataset: histograms of solutions of Graph-MIQUBO solved with Quantum Annealing (QA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded.

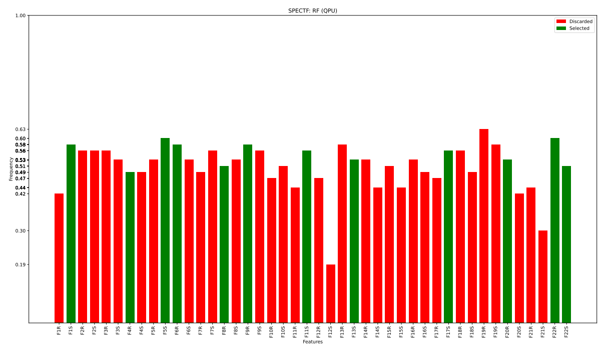
Furthermore, this gets more interesting by looking the feature frequency obtained by using Simulated Annealing: in fact, in Figures 4.9, 4.10 and 4.11, we can see the direct comparison between the two solvers' solutions on ”SPECTF” dataset, on the left column with Quantum Annealing (QA) and on the right with Simulated Annealing (SA). With respect to the previous comment, we can see that with Simulated Annealing, all three algorithms discard features below some frequency threshold, different for each one of them, with no exceptions. The shapes are very similar for Q-Correlation with both SA and QA, while for Graph-MIQUBO and Q-Boosting are different, even in the selection of the best features that provide the highest accuracy score over all the solutions found with k-combinations.

4.4.2 QUBO vs classic feature selection algorithms

The reported Table 4.10 (page 54) contains the results of the classifiers with the best accuracies obtained with Graph-MIQUBO, computed with Quantum Anneal-

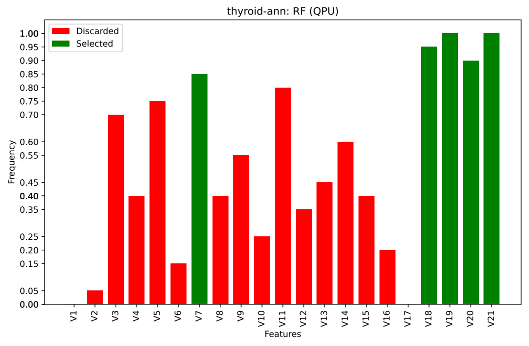


(a) Thyroid: Q-Boost (QA).

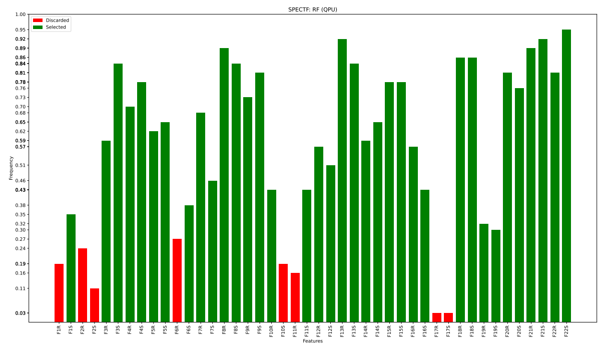


(b) SPECTF: Q-Boost (QA).

Figure 4.7: Thyroid vs SPECTF dataset: histograms of solutions of Q-Boosting solved with Quantum Annealing (QA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded.



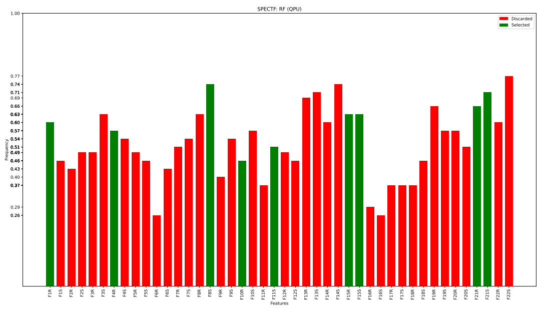
(a) Thyroid: Q-Corr (QA).



(b) SPECTF: Q-Corr (QA).

Figure 4.8: Thyroid vs SPECTF dataset: histograms of solutions of Q-Correlation solved with Quantum Annealing (QA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded.

ing, against the classical algorithms that have been explained in Section 2.2. Each method contains two separated columns labeled with N and CV_score: like the other


(a) SPECTF: G-MIQUBO (QA).

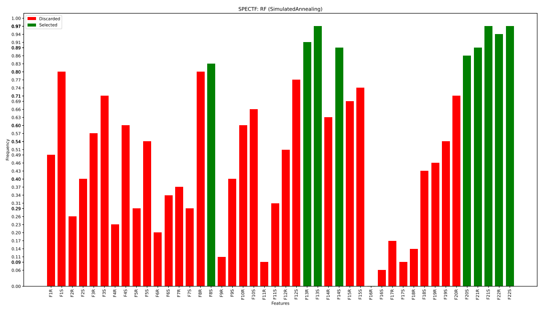
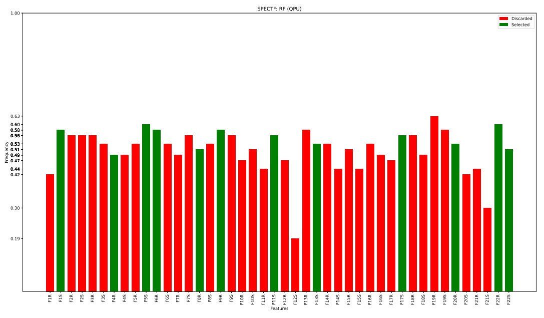

(b) SPECTF: G-MIQUBO (SA).

Figure 4.9: SPECTF dataset: histograms of solutions of Graph-MIQUBO solved with Quantum Annealing (QA) and Simulated Annealing (SA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded.


(a) SPECTF: Q-Boost (QA).

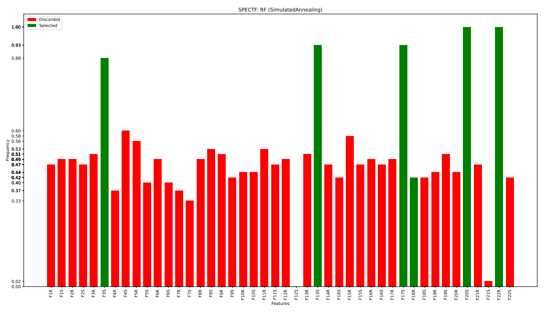
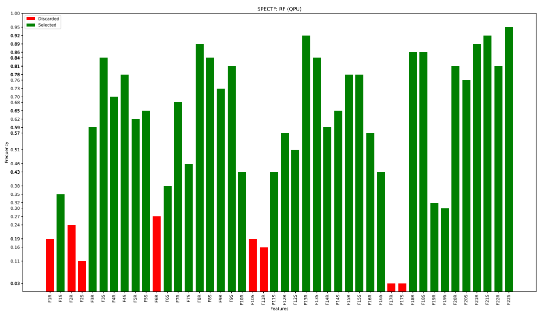

(b) SPECTF: Q-Boost (SA).

Figure 4.10: SPECTF dataset: histograms of solutions of Q-Boosting solved with Quantum Annealing (QA) and Simulated Annealing (SA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded.

tables that have been presented, N indicates the number of features of the model used to train the best classifier and CV_score the accuracy score, that we have called also *cross-validation score*.


(a) SPECTF: Q-Corr (QA).

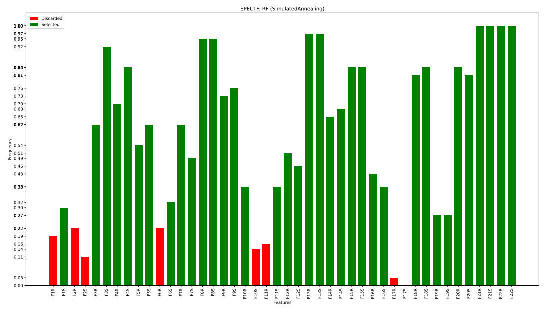

(b) SPECTF: Q-Corr (SA).

Figure 4.11: SPECTF dataset: histograms of solutions of Q-Correlation solved with Quantum Annealing (QA) and Simulated Annealing (SA) during the k-combinations, as features frequency. It shows the frequency of each feature being selected in the solutions of all k-combinations problems. The green bars correspond to the features selected of the best Random Forest found in validation phase, while the red bars correspond to the features discarded.

By looking at the mentioned table, it is possible to observe that the results across all the methods, both QUBO and classic, are comparable: generally, all methods tend to cut a consistent amount of features by increasing the accuracy with respect to the classifier trained with all the features of the original datasets.

A distinct behaviour of the QUBO method against the classic ones is not evident: with some datasets, Graph-MIQUBO solved with QA discards more features obtaining a smaller accuracy score, but this gap is generally very low, only less than 1% of difference between them. In other cases the classical algorithms tend to cut more features, often obtaining equal or higher accuracies, like with datasets "tecator" and "isolet".

Specifically for the Graph-MIQUBO algorithm, since its formulation is based on the statistical measure of *Mutual Information*, it is interesting to make a direct comparison with its classical version that does not include any QUBO formulation or quantum computing, but just a generated ranking of the features. The results are comparable, both algorithms provide a considerable cut of features and with improved accuracy from the starting learning problems.

Again, a general behaviour of this QUBO method against the classical ones cannot be distinctively highlighted. In this section only the table with Graph-MIQUBO has been commented. The tables with Q-Boosting and Q-Correlation results are reported in the Appendix, Section A.2.

	All Features		**Graph-MIQUBO [QPU]**		Variance Threshold		Mutual Information		Chi2 test		ANOVA f_test	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924 (0.0)	3	0.924	2	0.943	1	0.943	2	0.943
breast_cancer	30	0.957	**17**	**0.957 (0.0)**	29	0.957	21	0.960	22	0.960	24	0.960
wine	13	0.984	12	0.992 (0.0)	12	0.976	12	0.984	10	0.968	7	0.976
vehicle	18	0.753	**16**	**0.753 (0.0)**	17	0.747	17	0.752	16	0.742	17	0.743
ionosphere	34	0.918	32	0.931 (0.0)	33	0.931	18	0.935	17	0.931	22	0.935
robot-failures-lp5	90	0.702	70	0.754 (0.48)	50	0.745	80	0.763	83	0.746	78	0.737
waveform-5000	40	0.851	38	0.853 (0.07)	39	0.849	34	0.854	38	0.854	35	0.856
steel-plates-fault	33	0.989	32	0.982 (0.0)	32	0.935	31	0.991	26	0.995	18	0.995
nomao	118	0.966	92	0.966 (0.57)	50	0.965	86	0.967	98	0.966	95	0.966
SPECTF	44	0.790	9	0.817 (0.23)	43	0.806	21	0.833	8	0.828	23	0.822
cars1	7	0.799	6	0.803 (0.0)	6	0.770	1	0.854	5	0.799	2	0.858
LED-7digit	7	0.689	**6**	**0.717 (0.0)**	6	0.566	5	0.660	6	0.717	5	0.669
thyroid-ann	21	0.995	**12**	**0.997 (0.0)**	20	0.996	19	0.997	11	0.997	17	0.997
spambase	57	0.948	55	0.949 (0.05)	50	0.948	56	0.949	53	0.949	39	0.950
tecator	124	0.917	103	0.929 (0.57)	50	0.893	85	0.935	63	0.940	73	0.947
USPS	256	0.984	241	0.988	50	0.976	218	0.990	249	0.988	249	0.988
isolet	617	0.979	539	0.986	50	0.983	63	0.986	51	0.988	51	0.986
swarm-behaviour	2400	1.000	746	1.0	50	1.000	1811	1.000	1175	1.000	294	1.000

Table 4.10: Graph-MIQUBO solved with the QPU results comparison against classical filter feature selection algorithms. In the *cross-validation score* (CV_score) field of the QUBO method column, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach. Highlighted in bold the results where the Graph-MIQUBO feature selection method provided one of the highest number of features discarded, and better (or equal) score with respect to the classifiers trained with all features and trained using the solutions of the other algorithms.

4.5 Confusion matrices

Until this section, the results shown are obtained in the validation phase: this means that the true knowledge of the trained classifiers have not yet been tested with unseen data. The goal of the test phase is to visualize how the best classifiers trained on the subset of features selected by the algorithms perform with the test set of each dataset. This process unfolds the insights on what is called the generalization power of a trained classifier to classify correctly samples that have never been used in the training and validation steps of the machine learning pipeline.

As mentioned earlier, the criteria used to check the generalization capability of the reduced models is the confusion matrix. In this section, we present the discussion and analysis of some of the confusion matrices obtained with the experiments, with the main objective to check comparisons across the QUBO algorithms solved with Quantum Annealing, and also the results obtained with respect to the ones computed with classical feature selection algorithms.

The objective of this section is to show the test results obtained with some of the datasets that have been used for the experiments with all the algorithms described in this work, the complete list of confusion matrices for all the datasets, for all three QUBO methods (solved with Quantum Annealing), is reported in Section A.5 of the Appendix. Furthermore, it is possible for the reader to view the test results also with tables in Section A.6.

4.5.1 Test results on unbalanced dataset: breast cancer

An example result that we can consider is with the "breast_cancer" unbalanced dataset. As reported in previous sections, this particular dataset contains 30 features and has a target variable with two distinct classes, with a distribution of classes approximately at 37.2% and 62.8%. A total of 7 different confusion matrices are reported: in Figure 4.12 with three sub-figures each for a QUBO method, and in 4.13 with four confusion matrices, one for each classic method.

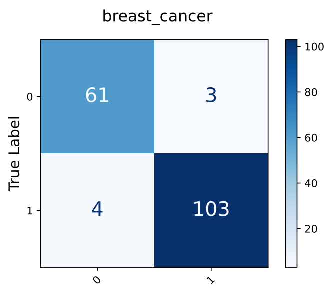
For the QUBO methods, the results are essentially the same, because in all three confusion matrices at least 95% of test samples are correctly classified. These are the results obtained considering the corresponding features selected and cross-validation score (Table 4.9, page 49):

- Graph-MIQUBO: 7 misclassified samples over 171 (95,9%), 17 features selected with accuracy (CV) equal to 0,957

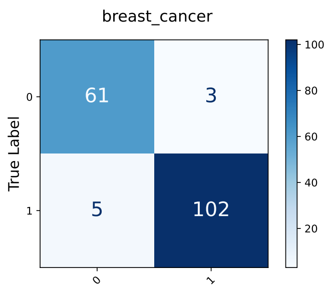
- Q-Boosting: 8 misclassified samples over 171 (95,3%), 25 features selected with accuracy (CV) equal to 0,96
- Q-Correlation: 6 misclassified samples over 171 (96,5%), 28 features selected with accuracy (CV) equal to 0,984

Despite the fact that Graph-MIQUBO has the lowest accuracy score (in validation) out of the three methods, in test phase the difference between them is very small, it is the method that discards nearly half of the starting 30 features.

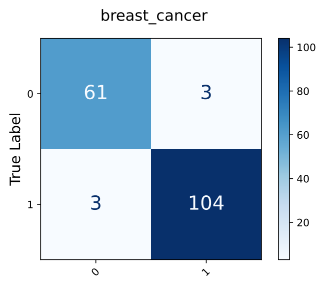
The results obtained with classical methods are similar: the four methods show a total of misclassifications between 4 to 8 samples. By looking back at the table with the performances, Graph-MIQUBO is still the one with the most features discarded with only 1,2% gap of accuracy in test phase. The closest to it in terms of features discarded is classic Mutual Information, which selects 21 features out of 30. Overall, we can conclude for this dataset that the performances are comparable, all with acceptable results in the test phase.



(a) Graph-MIQUBO



(b) Q-Boosting

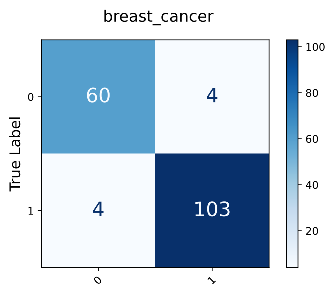


(c) Q-Correlation

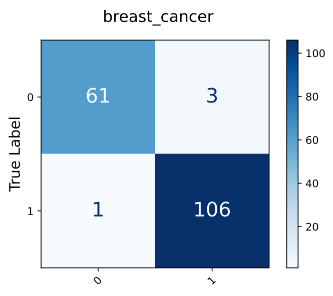
Figure 4.12: Confusion matrices on test set of breast cancer dataset with QUBO methods solved with the QPU.

4.5.2 Test results on balanced dataset: isolet

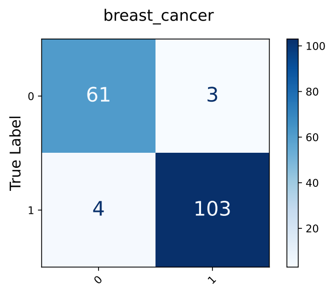
Here is reported an example on the results obtained in test phase for a balanced dataset. The ”isolet” dataset contains 617 features and 600 samples, 420 in training set and the remaining 180 for the test set. This dataset has two classes in the target variable and is balanced because the distribution of them corresponds to 50% and 50%. In Figures 4.14 and 4.15 the confusion matrices for the QUBO methods and classic ones are reported respectively. In the first one with QUBO methods, the three



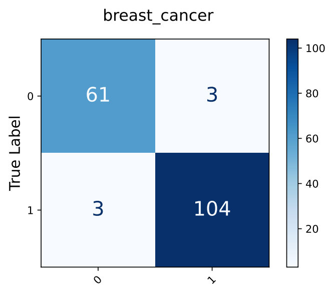
(a) Var. Thr.



(b) MI



(c) Chi2 Test



(d) ANOVA F-Test

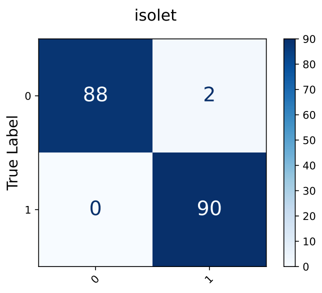
Figure 4.13: Confusion matrices on test set of breast cancer dataset with classic feature selection methods.

confusion matrices show that only 1 to 3 misclassifications only for one class out of 180 samples are made. By looking at the previous table with the QPU results, the method that discards most features is Q-Boosting, selecting only 114 out of 617, and provides in test phase only 3 misclassifications, with a test accuracy of 98,3%.

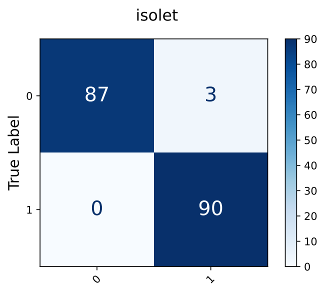
With classic methods, the results are again very similar: we can notice 2 to 4 total misclassifications across the four confusion matrices, scoring a test classification accuracy of at least 97,8%. This result indicates that all algorithms (both quantum-based and classical) obtain consistent test accuracies independently from the balance of the balance of presence of the classes in the target variable array.

4.6 Execution Times: comparison across the QUBO solvers

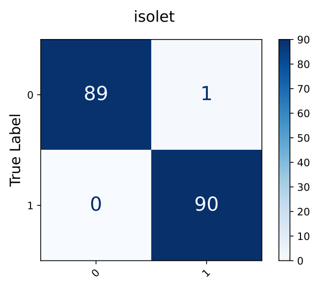
Another important metric is the *execution time* needed to compute all the previously shown results. In the following sections we provide another set of tables with



(a) Graph-MIQUBO

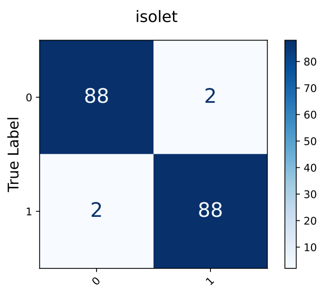


(b) Q-Boosting

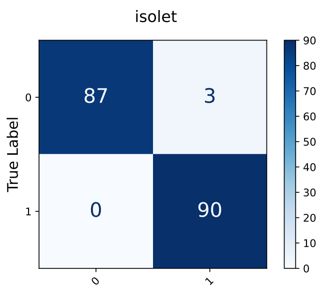


(c) Q-Correlation

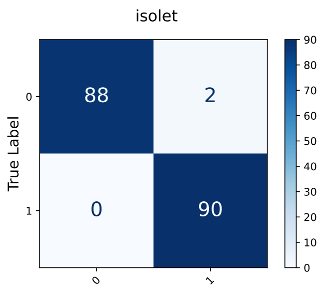
Figure 4.14: Confusion matrices on test set of isolet dataset with QUBO methods solved with the QPU.



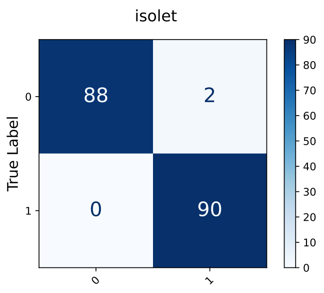
(a) Var. Thr.



(b) MI



(c) Chi2 Test



(d) ANOVA F-Test

Figure 4.15: Confusion matrices on test set of isolet dataset with classic feature selection methods.

the goal to understand the main differences about total execution time between the algorithms used for the experiments, both QUBO and classic, and also between the solving approaches used to compute the solutions of the optimization problems. For this current section, we focus on the comparison between the different solvers using the same QUBO algorithm.

Similarly to Section 4,3, a table for each QUBO method is provided, each containing the columns for the solvers implemented, reporting two different times:

- *QUBO generation* time: this field indicates the execution time needed to compute the square matrix that defines the objective function of the optimization problem.
- *Sampling* time: this is the total time needed for the corresponding solver to find the optimal solutions, so the ones with minimum energy, of the k-combinations. Thus, it is obtained with the sum of computational times needed to solve each k-combination problem.

The measurement of the sampling time has to be carefully considered when analysing the one obtained with Quantum Annealing: while the other approaches used to solve the optimization problem are locally ran by the machine, the quantum one involves the call to D-Wave's cloud service Leap. This difference implies that the overall time that passes from the call of the service to the moment in which the local system receives the response of the quantum annealer is influenced by multiple factors: the time needed to upload and send the model of the optimization problem to Leap, the inner queue to access the quantum machine and also the time needed by the calling machine to download and elaborate the response created. This is why in different tables a big variance of the measured sampling time is observed.

For all the tables with execution times that will be referenced, four datasets (”robot-failures-lp5”, ”nomao”, ”spambase” and ”tecator”) are not reported due to the conditions of the experiments performed over them. This includes slow and unstable internet connection, which makes the execution times, measured while using the Leap cloud service, unusable for a fair comparison.

In Table 4.11, the timings for the Graph-MIQUBO algorithm are reported. It is visible that, as the number of features grows, the time needed to compute the QUBO formulation increases: even if the square matrix used for the objective function is the same for each solver, for completeness the timings for each one have been reported, to have an overall idea of how much time is needed before starting the k-combinations. Furthermore, we can observe that the sampling times obtained with the quantum annealer are comparable or way less than the ones obtained with non-quantum heuristics, with some exceptions of very large times like with "nomao" and "tecator" datasets, which measurements are effected by the factors mentioned above when the Leap platform is used. In Tables 4.12 and 4.13, the timings of Q-

Graph-MIQUBO					
		QA	SA	TS	SD
Dataset	QUBO	Sampling	Sampling	Sampling	Sampling
iris	< 1s	< 1s	< 1s	01m 03s	< 1s
breast_cancer	2s	6s	13s	05m 36s	< 1s
wine	< 1s	< 1s	2s	03m 09s	< 1s
vehicle	< 1s	< 1s	6s	05m 36s	< 1s
ionosphere	2s	5s	25s	11m 33s	< 1s
waveform-5000	13s	03m 09s	48s	09m 06s	< 1s
steel-plates-fault	4s	01m 58s	34s	11m 12s	< 1s
SPECTF	4s	05m 41s	48s	12m 15s	< 1s
cars1	< 1s	< 1s	< 1s	01m 45s	< 1s
LED-7digit	< 1s	< 1s	< 1s	02m 06s	< 1s
thyroid-ann	2s	17s	9s	07m 00s	< 1s
USPS	03m 20s	01m 00s	01h 07m 53s	01h 02m 45s	53s
isolet	12m 51s	01m 40s	15h 21m 35s	04h 29m 20s	20m 49s
swarm-behaviour	08h 56m 03s	04m 09s	04h 10m 56s	02h 07m 28s	27m 00s

Table 4.11: Graph-MIQUBO execution times across the solvers. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems. Approaches used to solve QUBO problems: Quantum Annealing (QA), Simulated Annealing (SA), Tabu Search (TS) and Steepest Descent (SD).

Boosting and Q-Correlation across the solvers are reported respectively. As for the table for Graph-MIQUBO, the comments and analysis are similar: with some exceptions due to the Leap's cloud platform calls, Quantum Anneling beats all the solvers for sampling time when big datasets are used, while for smaller datasets Steepest Descent (SD) is always faster. Furthermore, regarding time measurements, SD and non-quantum approaches are more reliable since they are locally computed with no external factors that can influence their execution.

It is interesting to see the timings for the final three datasets, which leverage the hybrid quantum systems: with Hybrid-QPU the sampling times are not only much lower with respect to other solvers, but also equal or lower with respect to Steepest Descent (SD), which is the fastest non-quantum approach, due to its greedy policy as explained in the second chapter in Section 2.5.2.

Q-Boosting					
		QA	SA	TS	SD
Dataset	QUBO	Sampling	Sampling	Sampling	Sampling
iris	< 1s	< 1s	< 1s	42s	< 1s
breast_cancer	< 1s	01m 32s	13s	10m 09s	< 1s
wine	< 1s	2s	2s	03m 51s	< 1s
vehicle	< 1s	8s	6s	05m 57s	< 1s
ionosphere	< 1s	02m 17s	25s	11m 33s	< 1s
waveform-5000	< 1s	04m 56s	01m 12s	13m 39s	< 1s
steel-plates-fault	< 1s	02m 21s	47s	11m 12s	< 1s
SPECTF	< 1s	07m 15s	01m 05s	15m 03s	< 1s
cars1	< 1s	< 1s	< 1s	02m 06s	< 1s
LED-7digit	< 1s	< 1s	< 1s	02m 06s	< 1s
thyroid-ann	< 1s	17s	10s	06m 39s	< 1s
USPS	7s	01m 43s	55m 20s	01h 32m 21s	01m 20s
isolet	37s	01m 58s	03h 55m 15s	04h 39m 16s	21m 52s
swarm-behaviour	22m 45s	04m 32s	08h 40m 46s	02h 12m 43s	28m 28s

Table 4.12: Q-Boosting execution times across the solvers. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems. Approaches used to solve QUBO problems: Quantum Annealing (QA), Simulated Annealing (SA), Tabu Search (TS) and Steepest Descent (SD).

4.7 Execution Times: comparison across the algorithms

Following the same schema of Section 4.3 to Section 4.4 and explanation of Section 4.6, the opposite perspective to visualize the execution and sampling times is considered: the comparison of the timings between the QUBO algorithms solved with QPU and the one between QUBO and classic algorithms.

4.7.1 QUBO algorithms

In Table 4.14, we see the direct comparison across the QUBO algorithms, and a general pattern is visible: Graph-MIQUBO requires way more time to compute the formulation of the QUBO problem, while in general needs less time to perform all the sampling phases. Q-Boosting and Q-Correlation, on the other hand, have similar and comparable QUBO formulation and sampling times. This behaviour is expected due to the inner complexity of the formulas and terms used to obtain the square matrix that defines the optimization problem. This, specifically for Graph-MIQUBO,

Q-Correlation					
		QA	SA	TS	SD
Dataset	QUBO	Sampling	Sampling	Sampling	Sampling
iris	< 1s	< 1s	< 1s	01m 03s	< 1s
breast_cancer	< 1s	01m 17s	9s	09m 06s	< 1s
wine	< 1s	2s	2s	04m 12s	< 1s
vehicle	< 1s	9s	3s	05m 57s	< 1s
ionosphere	< 1s	02m 30s	11s	11m 33s	< 1s
waveform-5000	< 1s	04m 34s	38s	13m 39s	< 1s
steel-plates-fault	< 1s	02m 10s	20s	11m 12s	< 1s
SPECTF	< 1s	06m 50s	25s	12m 57s	< 1s
cars1	< 1s	< 1s	< 1s	02m 06s	< 1s
LED-7digit	< 1s	< 1s	< 1s	02m 06s	< 1s
thyroid-ann	< 1s	18s	8s	07m 00s	< 1s
USPS	7s	01m 55s	51m 21s	01h 31m 56s	01m 26s
isolet	36s	02m 18s	09h 49m 43s	04h 53m 49s	21m 25s
swarm-behaviour	20m 07s	05m 10s	13h 41m 18s	02h 16m 23s	28m 45s

Table 4.13: Q-Correlation execution times across the solvers. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems. Approaches used to solve QUBO problems: Quantum Annealing (QA), Simulated Annealing (SA), Tabu Search (TS) and Steepest Descent (SD).

becomes an issue for problems of large scale: in fact, with "swarm-behaviour", which is the biggest dataset out of all that have been used for the experiments, the difference of time needed to compute QUBO across the algorithms is tremendously high. As already mentioned is section 4.6, the *sampling time* measured using the QPU is affected by multiple factors, since the approach is not directly computed on the local machine, but via a cloud service. Then, that is why in the mentioned table there's a remarkable variance in the sampling time across all the algorithms.

4.7.2 QUBO vs classic feature selection algorithms

Finally, the last table that is relevant to discuss in this chapter is the one about timing measurements comparison between QUBO methods and classic ones. Similarly to Section 4.4.2, only Graph-MIQUBO is reported.The other two tables for Q-Boosting and Q-Correlation are inserted in Section A.4 of the Appendix.

The details about Graph-MIQUBO against the classical feature selection methods are in Table 4.15: as we have seen in previous sections, this QUBO method is the most

Quantum Annealing (QA)						
	Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	QUBO	Sampling	QUBO	Sampling	QUBO	Sampling
iris	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
breast_cancer	2s	6s	< 1s	01m 32s	< 1s	01m 17s
wine	< 1s	< 1s	< 1s	2s	< 1s	2s
vehicle	< 1s	< 1s	< 1s	8s	< 1s	9s
ionosphere	2s	5s	< 1s	02m 17s	< 1s	02m 30s
waveform-5000	13s	03m 09s	< 1s	04m 56s	< 1s	04m 34s
steel-plates-fault	4s	01m 58s	< 1s	02m 21s	< 1s	02m 10s
SPECTF	4s	05m 41s	< 1s	07m 15s	< 1s	06m 50s
cars1	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LED-7digit	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
thyroid-ann	2s	17s	< 1s	17s	< 1s	18s
USPS	03m 20s	01m 00s	7s	01m 43s	7s	01m 55s
isolet	12m 51s	01m 40s	37s	01m 58s	36s	02m 18s
swarm-behaviour	08h 56m 03s	04m 09s	22m 45s	04m 32s	20m 07s	05m 10s

Table 4.14: Quantum Annealing approach execution times across QUBO algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

computationally expensive out of all three to formulate the optimization problem, but the one with the lowest estimated sampling times. To make a fair comparison, we would need to consider the sum of the QUBO generation time and of the sampling time: that is because with classical feature selections there is no formulation of an optimization problem and no search of a minimum energy state phase, but rather a scoring computation given to all features and a greedy selection of the most promising. Following this schema, it is evident that some methods are faster than others: for instance, Variance Threshold (indicated as *VarThr*) is always the faster one. By recalling the formulas seen in Section 2.2, this classical method only computes the correlation between a feature and the target variable, while others have a higher inner mathematical complexity. The method with the highest complexity is Mutual Information (indicated with *Mutual Info*), because computing all the entropies that build the formula used to rank the features is more expensive to calculate.

It is interesting to compare Graph-MIQUBO and Mutual Information, because at the base of their implementation, both leverage the *Relevance* Formula (2.6) that has been introduced in Section 2.2.2. Despite this, there is a major difference between these two feature selection methods: given N features of a dataset, classic Mutual

Information has to compute N different values, one for each feature, that consist as the relevance of a feature f_i with respect to the target variable y, so $Rel(f_i, y)$. Graph-MIQUBO, on the other hand, has to compute all the possible combinations of relevances between the features themselves, thus making the starting computation of QUBO formulation much more expensive. Mind that for QUBO algorithms, this timing has to be then added to the sampling time needed for the solvers to obtain the final solutions.

The timing gap between this specific QUBO method and the classic ones gets intuitively more and more visible with larger datasets: with "swarm-behaviour" for instance, there is a striking total difference between Graph-MIQUBO (total time of 9 hours) and Variance Threshold (less than a second). Even if Q-Boosting and Q-Correlation are much faster to formulate the QUBO problem than Graph-MIQUBO, with a total time needed between 25 to 27 minutes, classical methods (except Mutual Information) are always much quicker to obtain a final solution and subset of features selected, due to inner simpler structure and overall implementation.

	Graph-MIQUBO [QPU]		Classic methods			
Dataset	QUBO	Sampling	VarThr	Mutual Info	Chi2	ANOVA
iris	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
breast_cancer	2s	6s	< 1s	2s	< 1s	< 1s
wine	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
vehicle	< 1s	< 1s	< 1s	1s	< 1s	< 1s
ionosphere	2s	5s	< 1s	2s	< 1s	< 1s
waveform-5000	13s	03m 09s	< 1s	21s	< 1s	< 1s
steel-plates-fault	4s	01m 58s	< 1s	6s	< 1s	< 1s
SPECTF	4s	05m 41s	< 1s	3s	< 1s	< 1s
cars1	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LED-7digit	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
thyroid-ann	2s	17s	< 1s	4s	< 1s	< 1s
USPS	03m 20s	01m 00s	< 1s	01m 04s	< 1s	< 1s
isolet	12m 51s	01m 40s	< 1s	01m 13s	< 1s	< 1s
swarm-behaviour	08h 56m 03s	04m 09s	< 1s	02h 33m 06s	13s	37s

Table 4.15: Graph-MIQUBO solved with the QPU execution times comparison against classical filter feature selection algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

Chapter 5

Conclusion

In this work, we have illustrated and explained filter feature selection algorithms that exploit quantum mechanical properties on specific machines. We decribed a graph based approach (*Graph Mutual Information QUBO*), *Quantum-Boosting* and *Quantum-Correlation*, which are all based on the formulation and generation of a QUBO model, and have all been tested with Quantum Annealing.

The results show the accuracies and timings obtained with quantum-based and classical algorithms. We have made detailed comparisons with different perspectives: the performance of a single QUBO algorithm with respect to different heuristics, including Quantum Annealing, and between all the algorithms themselves, given a solver for the QUBO ones.

We tested all the described algorithms on 18 datasets with different size, structure, and properties. We noticed that the accuracies of the Random Forests trained on the reduced datasets obtained with the methods, both quantum and non-quantum, are all comparable. By looking only at the QUBO algorithms, the results, in both validation and test phase, were similar across the heuristics used and slightly different across the algorithms themselves. While Graph-MIQUBO is the most expensive to formulate the QUBO problem, it is on average the fastest out of all three algorithms in solving the k-combinations problems.

Quantum-based algorithms solved with the quantum annealer, showed a consistent and comparable performance against the solutions found with classical algorithms. Furthermore, the analysed timings to solve a QUBO problem with Quantum Annealing showed an interesting scalability with respect to classical methods. However, the total execution time needed for QUBO feature selection methods is still higher compared with classical algorithms. So, the generation of the QUBO model becomes the bottleneck of the quantum-based methods.

Finally, we can conclude that the presented quantum-based algorithms solved with quantum annealer machines are capable of obtaining promising results with respect to the classic ones: Graph-MIQUBO is more indicated for datasets with smaller dimensionality, with remarkable reduction of the features used in the classification problem. Q-Boosting and Q-Correlation, on the other hand, are more indicated for problems with a larger set of features, since the generation of the QUBO model is much faster with respect to the graph-based one.

The results obtained are promising, considering that quantum annealer is relatively new technology. The next steps for this field of research is to improve and speed-up the time needed to compute a QUBO formulation, so to express deeper insights of interaction and relevance between the features themselves and with the target variable. Furthermore, a development of quantum annealers' technology could open up new possibilities for larger experiments that can be carried out without any hybrid approach.

Appendix A

All complete results

In this appendix section we are going to add all the tables, figures and results regarding the experiments performed and partially shown in the fourth chapter. All the following added contents are provided for completion to the reader.

A.1 Results: comparison across the algorithms

This section provides more complete results with respect to Section 4.4.1 regarding the comparison of the best classifiers obtained with a solver across the QUBO feature selection methods. In the referenced section only the results with the QPU are reported, here we provide the results obtained with:

- *Simulated Annealing*: Table A.1
- *Tabu Search*: Table A.2
- *Steepest Descent*: Table A.3

Simulated Annealing (SA)								
	All Features		Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.933	3	0.924	3	0.933
breast_cancer	30	0.957	19	0.962	25	0.960	29	0.960
wine	13	0.984	7	0.976	12	0.968	8	0.976
vehicle	18	0.753	17	0.747	17	0.748	16	0.747
ionosphere	34	0.918	32	0.935	19	0.939	23	0.931
robot-failures-lp5	90	0.702	74	0.746	66	0.746	77	0.745
waveform-5000	40	0.851	24	0.853	38	0.846	33	0.813
steel-plates-fault	33	0.989	32	0.983	32	0.992	29	0.992
nomao	118	0.966	106	0.967	103	0.967	85	0.967
SPECTF	44	0.790	9	0.817	6	0.844	36	0.817
cars1	7	0.799	5	0.814	5	0.810	5	0.814
LED-7digit	7	0.689	6	0.714	5	0.709	5	0.611
thyroid-ann	21	0.995	18	0.997	16	0.997	5	0.947
spambase	57	0.948	53	0.949	55	0.948	48	0.945
tecator	124	0.917	123	0.917	120	0.905	117	0.893
USPS	256	0.984	224	0.989	239	0.990	245	0.985
isolet	617	0.979	5	0.986	57	0.986	64	0.986
swarm-behaviour	2400	1.000	746	1.000	171	1.000	257	1.000

Table A.1: Simulated Annealing approach results across the algorithms.

Tabu Search (TS)								
	All Features		Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924	3	0.924	3	0.933
breast_cancer	30	0.957	15	0.957	22	0.960	27	0.957
wine	13	0.984	10	0.976	11	0.968	10	0.984
vehicle	18	0.753	16	0.748	17	0.748	13	0.745
ionosphere	34	0.918	32	0.935	18	0.943	25	0.927
robot-failures-lp5	90	0.702	80	0.746	77	0.755	74	0.771
waveform-5000	40	0.851	14	0.852	37	0.845	35	0.816
steel-plates-fault	33	0.989	32	0.982	10	0.998	29	0.991
nomao	118	0.966	105	0.966	116	0.967	90	0.967
SPECTF	44	0.790	33	0.822	5	0.844	14	0.806
cars1	7	0.799	5	0.811	4	0.807	5	0.825
LED-7digit	7	0.689	6	0.706	5	0.709	5	0.614
thyroid-ann	21	0.995	18	0.997	13	0.997	19	0.947
spambase	57	0.948	55	0.950	56	0.951	49	0.946
tecator	124	0.917	123	0.923	120	0.911	116	0.893
USPS	256	0.984	237	0.990	222	0.990	246	0.986
isolet	617	0.979	365	0.986	397	0.988	70	0.986
swarm-behaviour	2400	1.000	746	1.000	171	1.000	257	1.000

Table A.2: Tabu Search approach results across the algorithms.

Steepest Descent (SD)								
	All Features		Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924	3	0.924	3	0.933
breast_cancer	30	0.957	29	0.960	19	0.957	27	0.957
wine	13	0.984	10	0.976	11	0.976	8	0.968
vehicle	18	0.753	14	0.748	17	0.745	15	0.752
ionosphere	34	0.918	33	0.935	18	0.939	30	0.939
robot-failures-lp5	90	0.702	63	0.746	61	0.737	76	0.737
waveform-5000	40	0.851	19	0.854	37	0.845	35	0.820
steel-plates-fault	33	0.989	32	0.982	15	0.996	29	0.990
nomao	118	0.966	113	0.967	91	0.967	99	0.967
SPECTF	44	0.790	17	0.828	17	0.833	14	0.806
cars1	7	0.799	5	0.811	4	0.807	4	0.818
LED-7digit	7	0.689	6	0.706	5	0.706	5	0.609
thyroid-ann	21	0.995	19	0.997	14	0.996	19	0.947
spambase	57	0.948	56	0.949	48	0.948	53	0.945
tecator	124	0.917	123	0.881	120	0.917	116	0.899
USPS	256	0.984	240	0.990	230	0.988	245	0.986
isolet	617	0.979	473	0.986	38	0.986	33	0.986
swarm-behaviour	2400	1.000	746	1.000	171	1.000	257	1.000

Table A.3: Steepest Descent approach results across the algorithms.

A.2 Results: comparison QUBO vs classic feature selection algorithms

The following tables are additional results about the comparison between QUBO versus classical feature selection algorithms. In the dedicated Section 4.4.2, only the comparison with Graph-MIQUBO is provided. Here, the other tables for the missing QUBO algorithms are reported:

1. *Q-Boosting*: Table A.4
2. *Q-Correlation*: Table A.5

Notice that the columns corresponding to these two methods contain the results computed with Quantum Annealing, the other approaches to solve the optimization problem are not reported.

	All Features		**Q-Boosting [QPU]**		Variance Threshold		Mutual Information		Chi2 test		ANOVA f_test	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.924 (0.0)	3	0.924	2	0.943	1	0.943	2	0.943
breast_cancer	30	0.957	25	0.96 (0.0)	29	0.957	21	0.960	22	0.960	24	0.960
wine	13	0.984	11	0.976 (0.0)	12	0.976	12	0.984	10	0.968	7	0.976
vehicle	18	0.753	16	0.74 (0.0)	17	0.747	17	0.752	16	0.742	17	0.743
ionosphere	34	0.918	25	0.939 (0.06)	33	0.931	18	0.935	17	0.931	22	0.935
robot-failures-lp5	90	0.702	80	0.745 (0.31)	50	0.745	80	0.763	83	0.746	78	0.737
waveform-5000	40	0.851	32	0.854 (0.35)	39	0.849	34	0.854	38	0.854	35	0.856
steel-plates-fault	33	0.989	28	0.992 (0.09)	32	0.935	31	0.991	26	0.995	18	0.995
nomao	118	0.966	83	0.967 (0.65)	50	0.965	86	0.967	98	0.966	95	0.966
SPECTF	44	0.790	12	0.838 (0.32)	43	0.806	21	0.833	8	0.828	23	0.822
cars1	7	0.799	4	0.814 (0.0)	6	0.770	1	0.854	5	0.799	2	0.858
LED-7digit	7	0.689	5	0.709 (0.0)	6	0.566	5	0.660	6	0.717	5	0.669
thyroid-ann	21	0.995	11	0.997 (0.0)	20	0.996	19	0.997	11	0.997	17	0.997
spambase	57	0.948	40	0.949 (0.46)	50	0.948	56	0.949	53	0.949	39	0.950
tecator	124	0.917	120	0.905 (0.03)	50	0.893	85	0.935	63	0.940	73	0.947
USPS	256	0.984	238	0.987	50	0.976	218	0.990	249	0.988	249	0.988
isolet	617	0.979	114	0.986	50	0.983	63	0.986	51	0.988	51	0.986
swarm-behaviour	2400	1.000	171	1.0	50	1.000	1811	1.000	1175	1.000	294	1.000

Table A.4: Q-Boosting solved with the QPU results comparison against classical filter feature selection algorithms. In the *cross-validation score* (CV_score) field of the QUBO method column, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach.

	All Features		**Q-Correlation [QPU]**		Variance Threshold		Mutual Information		Chi2 test		ANOVA f_test	
Dataset	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score	N	CV_score
iris	4	0.924	3	0.933 (0.0)	3	0.924	2	0.943	1	0.943	2	0.943
breast_cancer	30	0.957	28	0.96 (0.0)	29	0.957	21	0.960	22	0.960	24	0.960
wine	13	0.984	10	0.984 (0.0)	12	0.976	12	0.984	10	0.968	7	0.976
vehicle	18	0.753	15	0.748 (0.0)	17	0.747	17	0.752	16	0.742	17	0.743
ionosphere	34	0.918	30	0.931 (0.03)	33	0.931	18	0.935	17	0.931	22	0.935
robot-failures-lp5	90	0.702	19	0.754 (0.39)	50	0.745	80	0.763	83	0.746	78	0.737
waveform-5000	40	0.851	35	0.818 (0.05)	39	0.849	34	0.854	38	0.854	35	0.856
steel-plates-fault	33	0.989	29	0.993 (0.0)	32	0.935	31	0.991	26	0.995	18	0.995
nomao	118	0.966	82	0.967 (0.69)	50	0.965	86	0.967	98	0.966	95	0.966
SPECTF	44	0.790	36	0.817 (0.02)	43	0.806	21	0.833	8	0.828	23	0.822
cars1	7	0.799	5	0.821 (0.0)	6	0.770	1	0.854	5	0.799	2	0.858
LED-7digit	7	0.689	5	0.603 (0.0)	6	0.566	5	0.660	6	0.717	5	0.669
thyroid-ann	21	0.995	5	0.946 (0.0)	20	0.996	19	0.997	11	0.997	17	0.997
spambase	57	0.948	51	0.948 (0.16)	50	0.948	56	0.949	53	0.949	39	0.950
tecator	124	0.917	117	0.881 (0.02)	50	0.893	85	0.935	63	0.940	73	0.947
USPS	256	0.984	245	0.983	50	0.976	218	0.990	249	0.988	249	0.988
isolet	617	0.979	174	0.986	50	0.983	63	0.986	51	0.988	51	0.986
swarm-behaviour	2400	1.000	257	1.0	50	1.000	1811	1.000	1175	1.000	294	1.000

Table A.5: Q-Correlation solved with the QPU results comparison against classical filter feature selection algorithms. In the *cross-validation score* (CV_score) field of the QUBO method column, is reported in parenthesis the *chain break fraction*, except for the last three datasets, which are solved with the hybrid approach.

A.3 Execution Times: comparison across the algorithms

This section provides more complete results with respect to Section 4.7.1 regarding the comparison of the execution and sampling times of the solvers across the QUBO feature selection methods. In the referenced section only the results with the QPU are reported. Here we provide the results obtained with:

- *Simulated Annealing*: Table A.6
- *Tabu Search*: Table A.7
- *Steepest Descent*: Table A.8

For all the referenced tables with execution times, four datasets (”robot-failures-lp5”, ”nomao”, ”spambase” and ”tecator”) are not reported due to the conditions of the experiments performed over them. This includes slow and unstable internet connection, which makes the execution times, measured while using the Leap cloud service, unusable for a fair comparison.

Simulated Annealing (SA)						
	Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	QUBO	Sampling	QUBO	Sampling	QUBO	Sampling
iris	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
breast_cancer	2s	13s	< 1s	13s	< 1s	9s
wine	< 1s	2s	< 1s	2s	< 1s	2s
vehicle	< 1s	6s	< 1s	6s	< 1s	3s
ionosphere	2s	25s	< 1s	25s	< 1s	11s
waveform-5000	13s	48s	< 1s	01m 12s	< 1s	38s
steel-plates-fault	4s	34s	< 1s	47s	< 1s	20s
SPECTF	4s	48s	< 1s	01m 05s	< 1s	25s
cars1	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LED-7digit	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
thyroid-ann	2s	9s	< 1s	10s	< 1s	8s
USPS	03m 20s	01h 07m 53s	7s	55m 20s	7s	51m 21s
isolet	12m 51s	15h 21m 35s	37s	03h 55m 15s	36s	09h 49m 43s
swarm-behaviour	08h 56m 03s	04h 10m 56s	22m 45s	08h 40m 46s	20m 07s	13h 41m 18s

Table A.6: Simulated Annealing approach execution times across QUBO algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

Tabu Search (TS)						
	Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	QUBO	Sampling	QUBO	Sampling	QUBO	Sampling
iris	< 1s	01m 03s	< 1s	42s	< 1s	01m 03s
breast_cancer	2s	05m 36s	< 1s	10m 09s	< 1s	09m 06s
wine	< 1s	03m 09s	< 1s	03m 51s	< 1s	04m 12s
vehicle	< 1s	05m 36s	< 1s	05m 57s	< 1s	05m 57s
ionosphere	2s	11m 33s	< 1s	11m 33s	< 1s	11m 33s
waveform-5000	13s	09m 06s	< 1s	13m 39s	< 1s	13m 39s
steel-plates-fault	4s	11m 12s	< 1s	11m 12s	< 1s	11m 12s
SPECTF	4s	12m 15s	< 1s	15m 03s	< 1s	12m 57s
cars1	< 1s	01m 45s	< 1s	02m 06s	< 1s	02m 06s
LED-7digit	< 1s	02m 06s	< 1s	02m 06s	< 1s	02m 06s
thyroid-ann	2s	07m 00s	< 1s	06m 39s	< 1s	07m 00s
USPS	03m 20s	01h 02m 45s	7s	01h 32m 21s	7s	01h 31m 56s
isolet	12m 51s	04h 29m 20s	37s	04h 39m 16s	36s	04h 53m 49s
swarm-behaviour	08h 56m 03s	02h 07m 28s	22m 45s	02h 12m 43s	20m 07s	02h 16m 23s

Table A.7: Tabu Search approach execution times across QUBO algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

Steepest Descent (SD)						
	Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	QUBO	Sampling	QUBO	Sampling	QUBO	Sampling
iris	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
breast_cancer	2s	< 1s	< 1s	< 1s	< 1s	< 1s
wine	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
vehicle	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
ionosphere	2s	< 1s	< 1s	< 1s	< 1s	< 1s
waveform-5000	13s	< 1s	< 1s	< 1s	< 1s	< 1s
steel-plates-fault	4s	< 1s	< 1s	< 1s	< 1s	< 1s
SPECTF	4s	< 1s	< 1s	< 1s	< 1s	< 1s
cars1	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LED-7digit	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
thyroid-ann	2s	< 1s	< 1s	< 1s	< 1s	< 1s
USPS	03m 20s	53s	7s	01m 20s	7s	01m 26s
isolet	12m 51s	20m 49s	37s	21m 52s	36s	21m 25s
swarm-behaviour	08h 56m 03s	27m 00s	22m 45s	28m 28s	20m 07s	28m 45s

Table A.8: Steepest Descent approach execution times across QUBO algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

A.4 Execution Times: comparison QUBO vs classic feature selection algorithms

The following tables are additional results about the comparison of execution and sampling times between QUBO and classical feature selection algorithms. In the dedicated Section 4.7.2, only the comparison with Graph-MIQUBO is provided. Here, the other tables for the missing QUBO algorithms are reported:

1. *Q-Boosting*: Table A.9
2. *Q-Correlation*: Table A.10

Notice that the columns corresponding to these two methods contain the results computed with Quantum Annealing, the other approaches to solve the optimization problem are not reported.

For all the referenced tables with execution times, four datasets (”robot-failures-lp5”, ”nomao”, ”spambase” and ”tecator”) are not reported due to the conditions of the experiments performed over them. This includes slow and unstable internet connection, which makes the execution times, measured while using the Leap cloud service, unusable for a fair comparison.

	Q-Boosting [QPU]		Classic methods			
Dataset	QUBO	Sampling	VarThr	Mutual Info	Chi2	ANOVA
iris	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
breast_cancer	< 1s	01m 32s	< 1s	2s	< 1s	< 1s
wine	< 1s	2s	< 1s	< 1s	< 1s	< 1s
vehicle	< 1s	8s	< 1s	1s	< 1s	< 1s
ionosphere	< 1s	02m 17s	< 1s	2s	< 1s	< 1s
waveform-5000	< 1s	04m 56s	< 1s	21s	< 1s	< 1s
steel-plates-fault	< 1s	02m 21s	< 1s	6s	< 1s	< 1s
SPECTF	< 1s	07m 15s	< 1s	3s	< 1s	< 1s
cars1	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LED-7digit	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
thyroid-ann	< 1s	17s	< 1s	4s	< 1s	< 1s
USPS	7s	01m 43s	< 1s	01m 04s	< 1s	< 1s
isolet	37s	01m 58s	< 1s	01m 13s	< 1s	< 1s
swarm-behaviour	22m 45s	04m 32s	< 1s	02h 33m 06s	13s	37s

Table A.9: Q-Boosting solved with the QPU execution times comparison against classical feature selection algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

	Q-Correlation [QPU]		Classic methods			
Dataset	QUBO	Sampling	VarThr	Mutual Info	Chi2	ANOVA
iris	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
breast_cancer	< 1s	01m 17s	< 1s	2s	< 1s	< 1s
wine	< 1s	2s	< 1s	< 1s	< 1s	< 1s
vehicle	< 1s	9s	< 1s	1s	< 1s	< 1s
ionosphere	< 1s	02m 30s	< 1s	2s	< 1s	< 1s
waveform-5000	< 1s	04m 34s	< 1s	21s	< 1s	< 1s
steel-plates-fault	< 1s	02m 10s	< 1s	6s	< 1s	< 1s
SPECTF	< 1s	06m 50s	< 1s	3s	< 1s	< 1s
cars1	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LED-7digit	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
thyroid-ann	< 1s	18s	< 1s	4s	< 1s	< 1s
USPS	7s	01m 55s	< 1s	01m 04s	< 1s	< 1s
isolet	36s	02m 18s	< 1s	01m 13s	< 1s	< 1s
swarm-behaviour	20m 07s	05m 10s	< 1s	02h 33m 06s	13s	37s

Table A.10: Q-Correlation solved with the QPU execution times comparison against classical feature selection algorithms. The "QUBO" field reports the time needed to formulate the QUBO problem. The "Sampling" field indicates what we called the *sampling time*, which is the total time needed by the solver to find the solutions with minimum energy for all k-combinations problems.

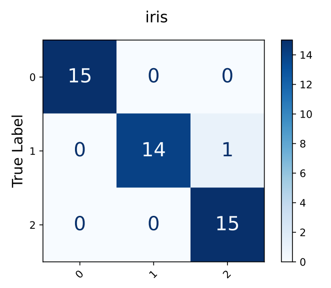
A.5 Confusion Matrices

In this section, the confusion matrices for all datasets are reported. Since there is a huge number of combinations between the algorithms, both from QUBO and classic categories, and all the solvers, the confusion matrices that are reported are only for the QUBO feature selection methods solved with Quantum Annealing heuristic, so with the QPU, for all datasets. Following the same definition used in Section 4.5, the results are obtained with the best trained Random Forest over the test set of each dataset.

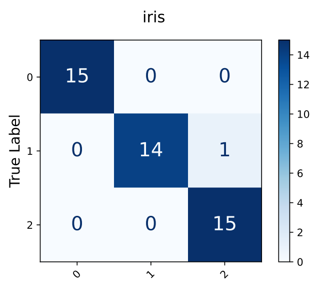
The following 6 pages (from page 82 to 87) contain the confusion matrices, each with a 3×3 schema: each row is a dataset, and each column corresponds to a QUBO method (from left to right *Graph-MIQUBO*, *Q-Boosting* and *Q-Correlation*).

The pages are divided as follows:

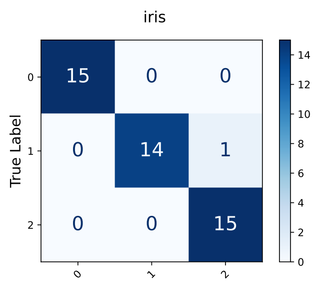
1. Page 82: *iris*, *breast-cancer* and *wine* datasets.
2. Page 83: *vehicle*, *ionosphere* and *robot-failures-lp5* datasets.
3. Page 84: *waveform-5000*, *steel-plates-fault* and *nomao* datasets.
4. Page 85: *SPECTF*, *cars1* and *LED-7digit* datasets.
5. Page 86: *thyroid-ann*, *spambase* and *tecator* datasets.
6. Page 87: *USPS*, *isolet* and *swarm-behaviour* datasets. Mind that for this page the results are obtained using the Hybrid QPU, since all these datasets have more than 124 features.



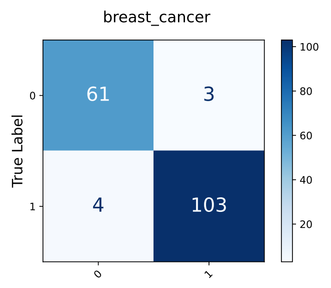
(a) iris: G-MIQUBO.



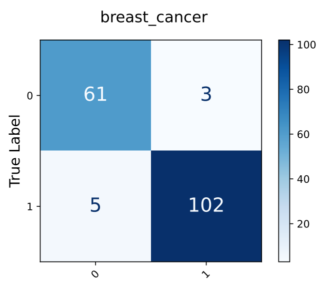
(b) iris: Q-Boost.



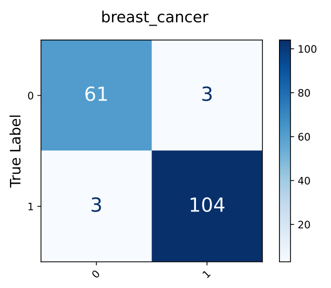
(c) iris: Q-Corr.



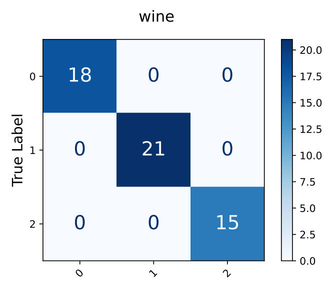
(d) breast-cancer: G-MIQUBO.



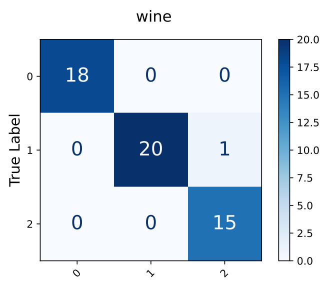
(e) breast-cancer: Q-Boost.



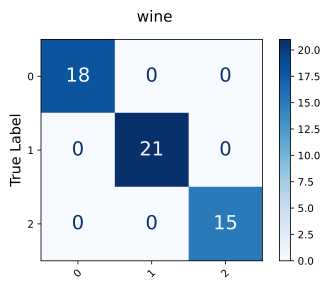
(f) breast-cancer: Q-Corr.



(g) wine: G-MIQUBO.

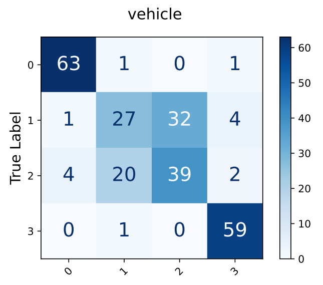


(h) wine: Q-Boost.

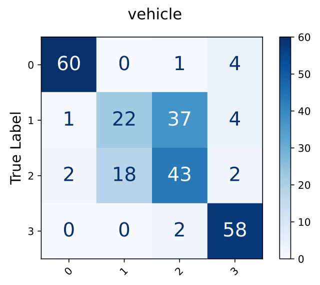


(i) wine: Q-Corr.

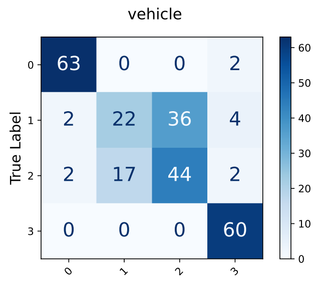
Figure A.1: All confusion matrices on test sets for all datasets with only the best subset of features selected with the QUBO algorithms solved with the QPU. Part 1 of 6.



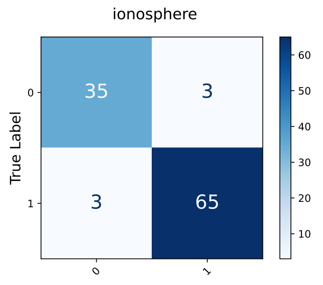
(a) vehicle: G-MIQUBO.



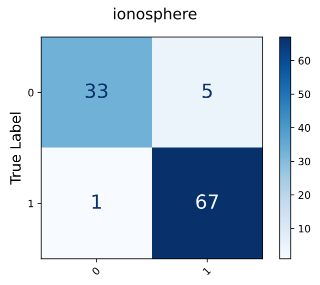
(b) vehicle: Q-Boost.



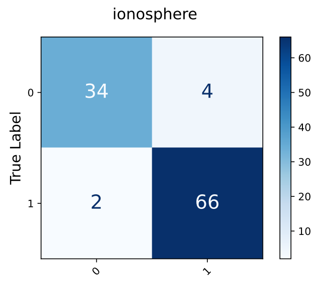
(c) vehicle: Q-Corr.



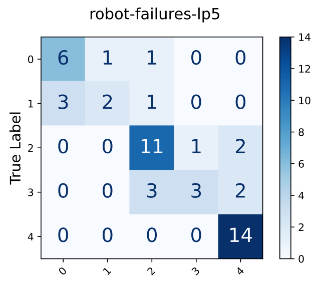
(d) ionosphere: G-MIQUBO.



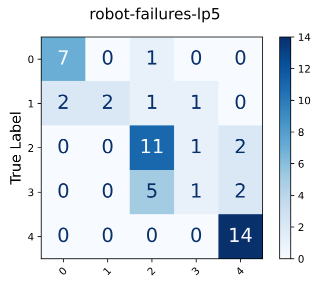
(e) ionosphere: Q-Boost.



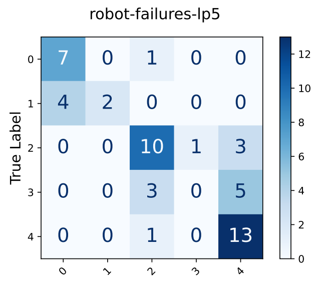
(f) ionosphere: Q-Corr.



(g) robot-f-lp5: G-MIQUBO.

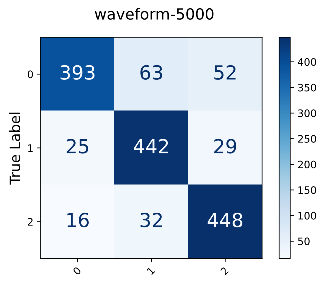


(h) robot-f-lp5: Q-Boost.

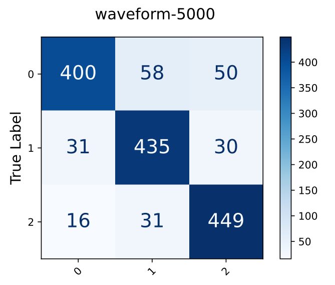


(i) robot-f-lp5: Q-Corr.

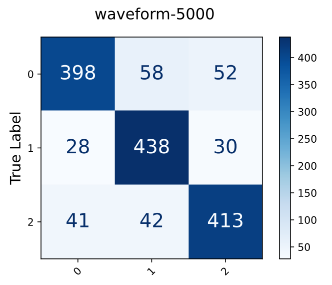
Figure A.2: All confusion matrices on test sets for all datasets with only the best subset of features selected with the QUBO algorithms solved with the QPU. Part 2 of 6.



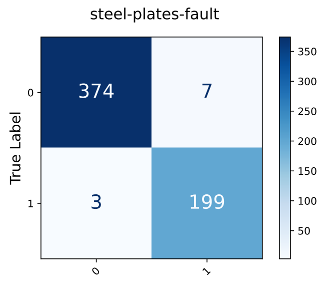
(a) waveform: G-MIQUBO.



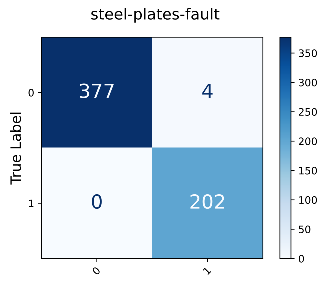
(b) waveform: Q-Boost.



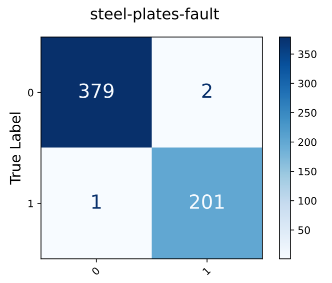
(c) waveform: Q-Corr.



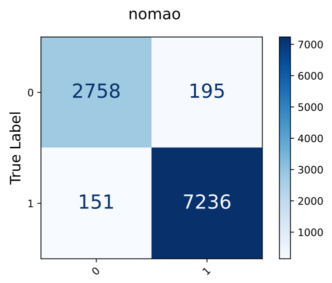
(d) steel-p-f: G-MIQUBO.



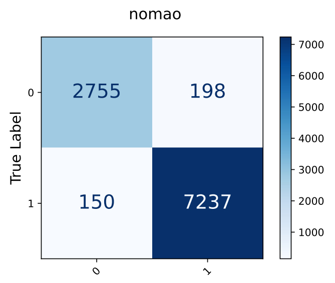
(e) steel-p-f: Q-Boost.



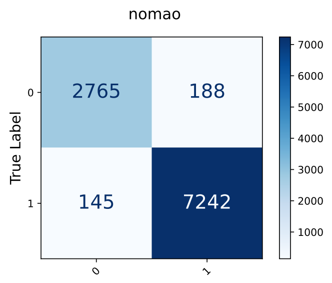
(f) steel-p-f: Q-Corr.



(g) nomao: G-MIQUBO.

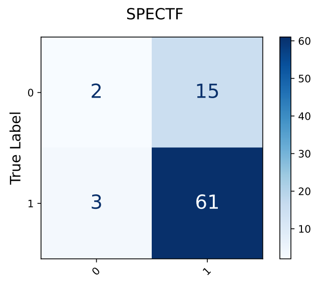


(h) nomao: Q-Boost.

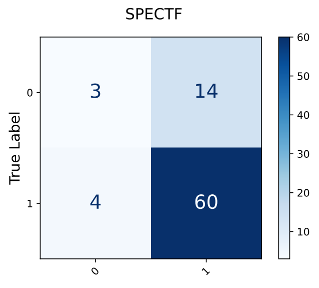


(i) nomao: Q-Corr.

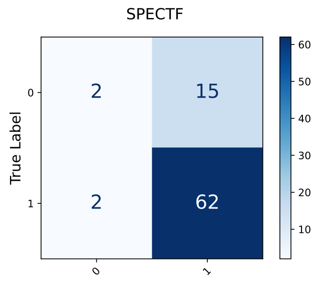
Figure A.3: All confusion matrices on test sets for all datasets with only the best subset of features selected with the QUBO algorithms solved with the QPU. Part 3 of 6.



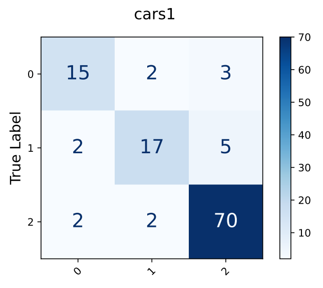
(a) SPECTF: G-MIQUBO.



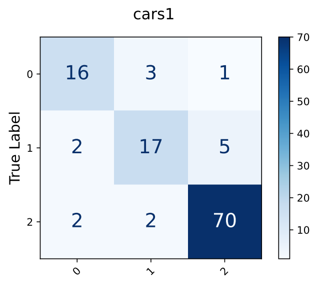
(b) SPECTF: Q-Boost.



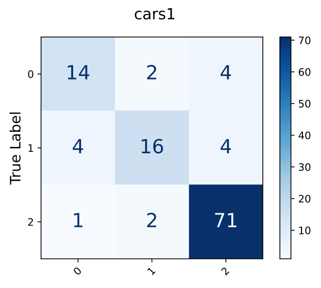
(c) SPECTF: Q-Corr.



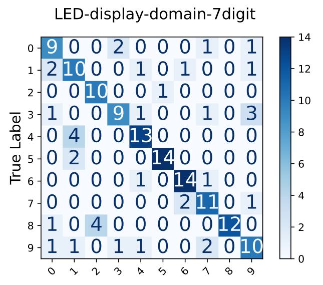
(d) cars1: G-MIQUBO.



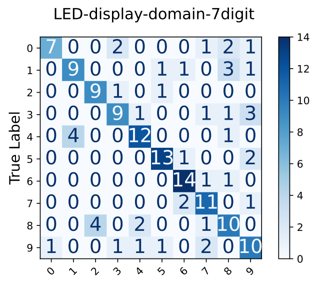
(e) cars1: Q-Boost.



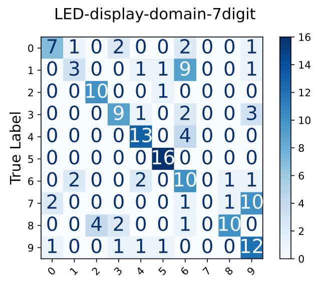
(f) cars1: Q-Corr.



(g) LED-7digit: G-MIQUBO.

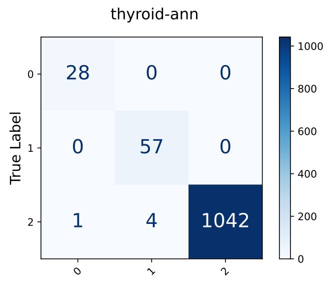


(h) LED-7digit: Q-Boost.

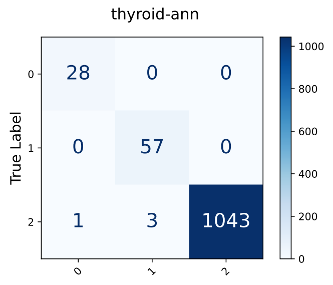


(i) LED-7digit: Q-Corr.

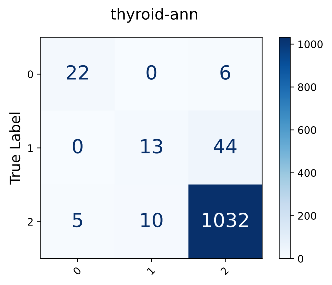
Figure A.4: All confusion matrices on test sets for all datasets with only the best subset of features selected with the QUBO algorithms solved with the QPU. Part 4 of 6.



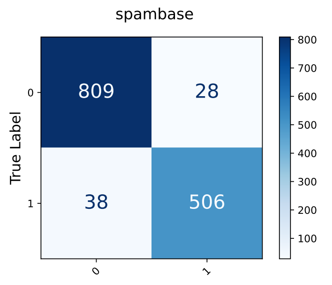
(a) thyroid-ann: G-MIQUBO.



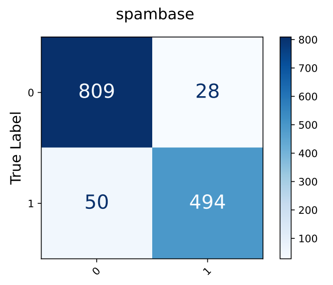
(b) thyroid-ann: Q-Boost.



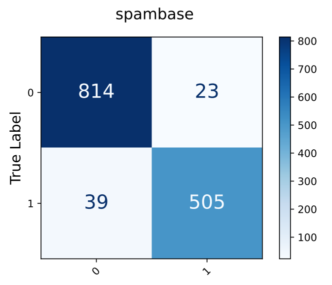
(c) thyroid-ann: Q-Corr.



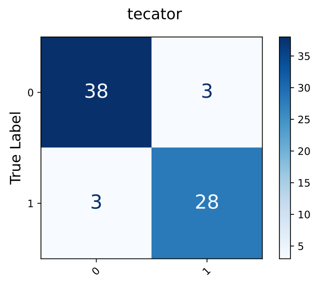
(d) spambase: G-MIQUBO.



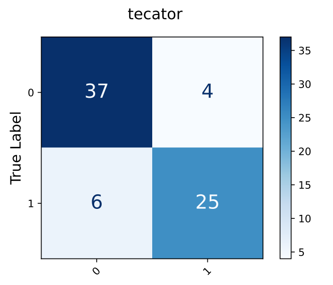
(e) spambase: Q-Boost.



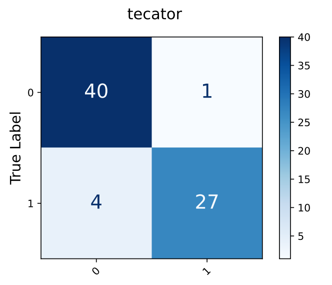
(f) spambase: Q-Corr.



(g) tecator: G-MIQUBO.

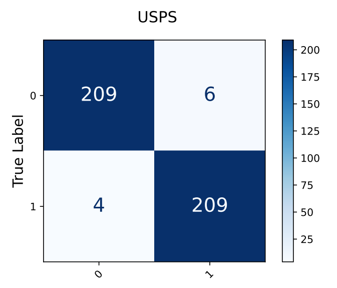


(h) tecator: Q-Boost.

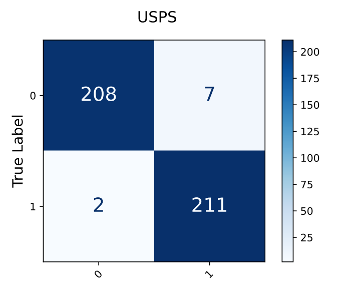


(i) tecator: Q-Corr.

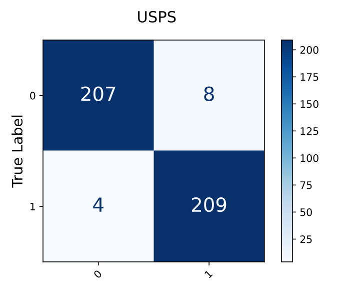
Figure A.5: All confusion matrices on test sets for all datasets with only the best subset of features selected with the QUBO algorithms solved with the QPU. Part 5 of 6.



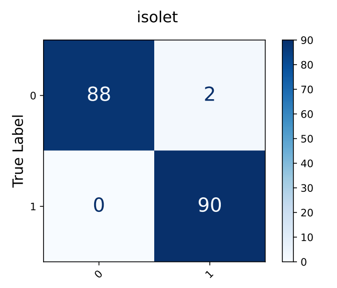
(a) USPS: G-MIQUBO.



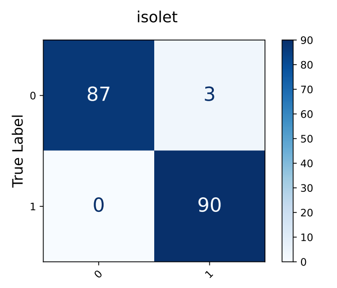
(b) USPS: Q-Boost.



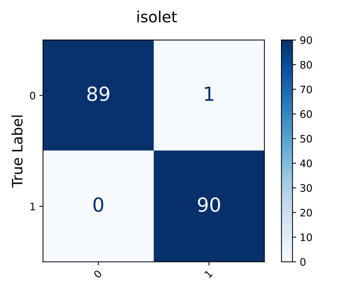
(c) USPS: Q-Corr.



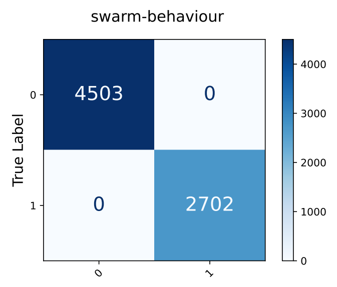
(d) isolet: G-MIQUBO.



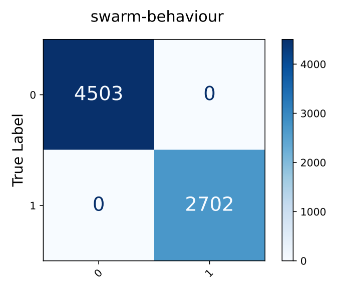
(e) isolet: Q-Boost.



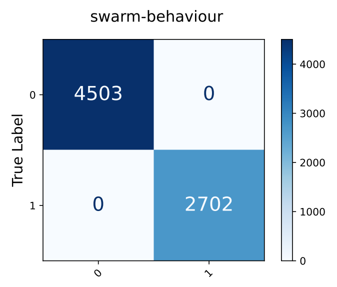
(f) isolet: Q-Corr.



(g) swarm: G-MIQUBO.



(h) swarm: Q-Boost.



(i) swarm: Q-Corr.

Figure A.6: All confusion matrices on test sets for all datasets with only the best subset of features selected with the QUBO algorithms solved with the QPU. Part 6 of 6. The three reported datasets have more than 124 features, so the hybrid approach has to be adopted.

A.6 Test classification accuracies: tables

The following tables are additional results obtained in test phase about the comparison of the QUBO (quantum) based features selection methods between themselves, and the comparison between QUBO versus classical feature selection algorithms. In the dedicated Section 4.5 (with the additional images in Section A.5), only the comparison with the confusion matrices is provided. Here, the tables for all datasets are reported:

1. Table A.11 (page 89): comparison across QUBO feature selection methods of the classification accuracies obtained with the best classifier found in validation phase using the unseen **test** set of each dataset.

2. Table A.12 (page 90): comparison between *Graph-MIQUBO* and classical feature selection algorithms of classification accuracies using the **test** set of each dataset.

3. Table A.13 (page 91): comparison between *Q-Boosting* and classical feature selection algorithms of classification accuracies using the **test** set of each dataset.

4. Table A.14 (page 92): comparison between *Q-Correlation* and classical feature selection algorithms of classification accuracies using the **test** set of each dataset.

Notice that the columns corresponding to the QUBO feature selection methods for the last three tables contain the results computed with Quantum Annealing, the other approaches to solve the optimization problem are not reported.

Quantum Annealing (QA)								
	All Features		Graph-MIQUBO		Q-Boosting		Q-Correlation	
Dataset	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.
iris	4	97.8%	3	97.8%	3	97.8%	3	97.8%
breast_cancer	30	95.9%	17	95.9%	25	95.3%	28	96.5%
wine	13	100.0%	12	100.0%	11	98.1%	10	100.0%
vehicle	18	77.2%	16	74.0%	16	72.0%	15	74.4%
ionosphere	34	94.3%	32	94.3%	25	94.3%	30	94.3%
robot-failures-lp5	90	72.0%	70	72.0%	80	70.0%	19	64.0%
waveform-5000	40	85.4%	38	85.5%	32	85.6%	35	83.3%
steel-plates-fault	33	99.7%	32	98.3%	28	99.3%	29	99.5%
nomao	118	96.8%	92	96.7%	83	96.6%	82	96.8%
SPECTF	44	79.0%	9	77.8%	12	77.8%	36	79.0%
cars1	7	87.3%	6	86.4%	4	87.3%	5	85.6%
LED-7digit	7	74.0%	6	74.7%	5	69.3%	5	60.0%
thyroid-ann	21	99.4%	12	99.6%	11	99.6%	5	94.3%
spambase	57	95.7%	55	95.2%	40	94.4%	51	95.5%
tecator	124	90.3%	103	91.7%	120	86.1%	117	93.1%
USPS	256	97.7%	241	97.7%	238	97.9%	245	97.2%
isolet	617	98.9%	539	98.9%	114	98.3%	174	99.4%
swarm-behaviour	2400	100.0%	746	100.0%	171	100.0%	257	100.0%

Table A.11: Quantum Annealing approach **test** results across the algorithms. Only for the last three datasets, the hybrid approach has to be adopted. In the "Test Acc." field is reported as percentage the classification accuracy with the best classifiers over the test set of each dataset.

	All Features		**Graph-MIQUBO [QPU]**		Variance Threshold		Mutual Information		Chi2 test		ANOVA f_test	
Dataset	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.
iris	4	97.8%	3	97.8%	3	95.6%	2	97.8%	1	93.3%	2	95.6%
breast_cancer	30	95.9%	17	95.9%	29	95.9%	21	95.9%	22	96.5%	24	95.9%
wine	13	100.0%	12	100.0%	12	100.0%	12	100.0%	10	100.0%	7	100.0%
vehicle	18	77.2%	16	74.0%	17	78.0%	17	74.4%	16	77.2%	17	71.7%
ionosphere	34	94.3%	32	94.3%	33	94.3%	18	95.3%	17	96.2%	22	95.3%
robot-failures-lp5	90	72.0%	70	72.0%	50	68.0%	80	72.0%	83	76.0%	78	74.0%
waveform-5000	40	85.4%	38	85.5%	39	83.8%	34	85.5%	38	85.3%	35	85.7%
steel-plates-fault	33	99.7%	32	98.3%	32	93.0%	31	99.0%	26	99.8%	18	99.8%
nomao	118	96.8%	92	96.7%	50	96.6%	86	96.7%	98	96.7%	95	96.7%
SPECTF	44	79.0%	9	77.8%	43	81.5%	21	80.2%	8	77.8%	23	80.2%
cars1	7	87.3%	6	86.4%	6	85.6%	1	83.1%	5	87.3%	2	83.1%
LED-7digit	7	74.0%	6	74.7%	6	62.0%	5	66.0%	6	72.7%	5	65.3%
thyroid-ann	21	99.4%	12	99.6%	20	99.5%	19	99.3%	11	99.6%	17	99.6%
spambase	57	95.7%	55	95.2%	50	95.4%	56	95.8%	53	95.3%	39	95.4%
tecator	124	90.3%	103	91.7%	50	91.7%	85	93.1%	63	90.3%	73	91.7%
USPS	256	97.7%	241	97.7%	50	96.5%	218	97.7%	249	97.7%	249	97.4%
isolet	617	98.9%	539	98.9%	50	98.3%	63	98.3%	51	98.9%	51	98.3%
swarm-behaviour	2400	100.0%	746	100.0%	50	100.0%	1811	100.0%	1175	100.0%	294	100.0%

Table A.12: Graph-MIQUBO solved with the QPU **test** results comparison against classical filter feature selection algorithms. Only for the last three datasets, the hybrid approach has to be adopted. In the "Test Acc." field is reported as percentage the classification accuracy with the best classifiers over the test set of each dataset.

	All Features		**Q-Boosting [QPU]**		Variance Threshold		Mutual Information		Chi2 test		ANOVA f_test	
Dataset	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.
iris	4	97.8%	3	97.8%	3	95.6%	2	95.6%	1	93.3%	2	97.8%
breast_cancer	30	95.9%	25	95.3%	29	95.9%	21	96.5%	22	96.5%	24	96.5%
wine	13	100.0%	11	98.1%	12	100.0%	12	100.0%	10	100.0%	7	100.0%
vehicle	18	77.2%	16	72.0%	17	75.6%	17	73.6%	16	74.8%	17	75.2%
ionosphere	34	94.3%	25	94.3%	33	95.3%	18	95.3%	17	95.3%	22	94.3%
robot-failures-lp5	90	72.0%	80	70.0%	50	68.0%	80	72.0%	83	72.0%	78	70.0%
waveform-5000	40	85.4%	32	85.6%	39	84.0%	34	85.1%	38	85.9%	35	85.4%
steel-plates-fault	33	99.7%	28	99.3%	32	93.8%	31	99.5%	26	99.5%	18	100.0%
nomao	118	96.8%	83	96.6%	50	96.5%	86	96.7%	98	96.7%	95	96.7%
SPECTF	44	79.0%	12	77.8%	43	80.2%	21	80.2%	8	77.8%	23	80.2%
cars1	7	87.3%	4	87.3%	6	85.6%	1	82.2%	5	87.3%	2	83.9%
LED-7digit	7	74.0%	5	69.3%	6	61.3%	5	66.7%	6	71.3%	5	66.0%
thyroid-ann	21	99.4%	11	99.6%	20	99.4%	19	99.5%	11	99.5%	17	99.6%
spambase	57	95.7%	40	94.4%	50	95.6%	56	95.4%	53	95.7%	39	95.1%
tecator	124	90.3%	120	86.1%	50	91.7%	85	93.1%	63	90.3%	73	94.4%
USPS	256	97.7%	238	97.9%	50	96.0%	218	97.9%	249	97.7%	249	97.4%
isolet	617	98.9%	114	98.3%	50	98.3%	63	98.3%	51	98.9%	51	98.3%
swarm-behaviour	2400	100.0%	171	100.0%	50	100.0%	1811	100.0%	1175	100.0%	294	100.0%

Table A.13: Q-Boosting solved with the QPU **test** results comparison against classical filter feature selection algorithms. Only for the last three datasets, the hybrid approach has to be adopted. In the "Test Acc." field is reported as percentage the classification accuracy with the best classifiers over the test set of each dataset.

	All Features		**Q-Correlation [QPU]**		Variance Threshold		Mutual Information		Chi2 test		ANOVA f_test	
Dataset	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.	N	Test Acc.
iris	4	97.8%	3	97.8%	3	95.6%	2	95.6%	1	93.3%	2	97.8%
breast_cancer	30	95.9%	28	96.5%	29	95.3%	21	97.7%	22	95.9%	24	96.5%
wine	13	100.0%	10	100.0%	12	100.0%	12	100.0%	10	100.0%	7	100.0%
vehicle	18	77.2%	15	74.4%	17	76.0%	17	75.2%	16	73.2%	17	73.2%
ionosphere	34	94.3%	30	94.3%	33	94.3%	18	94.3%	17	94.3%	22	95.3%
robot-failures-lp5	90	72.0%	19	64.0%	50	70.0%	80	72.0%	83	70.0%	78	70.0%
waveform-5000	40	85.4%	35	83.3%	39	84.4%	34	86.0%	38	85.7%	35	85.6%
steel-plates-fault	33	99.7%	29	99.5%	32	93.0%	31	99.5%	26	99.8%	18	99.8%
nomao	118	96.8%	82	96.8%	50	96.6%	86	96.7%	98	96.7%	95	96.7%
SPECTF	44	79.0%	36	79.0%	43	79.0%	21	80.2%	8	76.5%	23	79.0%
cars1	7	87.3%	5	85.6%	6	85.6%	1	82.2%	5	88.1%	2	83.1%
LED-7digit	7	74.0%	5	60.0%	6	58.7%	5	66.0%	6	72.7%	5	66.0%
thyroid-ann	21	99.4%	5	94.3%	20	99.2%	19	99.4%	11	99.6%	17	99.6%
spambase	57	95.7%	51	95.5%	50	95.4%	56	95.1%	53	95.6%	39	95.3%
tecator	124	90.3%	117	93.1%	50	93.1%	85	93.1%	63	93.1%	73	93.1%
USPS	256	97.7%	245	97.2%	50	97.0%	218	97.7%	249	97.7%	249	97.7%
isolet	617	98.9%	174	99.4%	50	97.8%	63	98.3%	51	98.9%	51	98.9%
swarm-behaviour	2400	100.0%	257	100.0%	50	100.0%	1811	100.0%	1175	100.0%	294	100.0%

Table A.14: Q-Correlation solved with the QPU **test** results comparison against classical filter feature selection algorithms. Only for the last three datasets, the hybrid approach has to be adopted. In the "Test Acc." field is reported as percentage the classification accuracy with the best classifiers over the test set of each dataset.

Bibliography

[1] N. Bogunović A. Jović K. Brkić. "A review of feature selection methods with applications". In: (). URL: `http://161.53.22.65/datoteka/763354.MIPRO_2015_JovicBrkicBogunovic.pdf`.

[2] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[3] Jacob Benesty et al. "Pearson correlation coefficient". In: (2009), pp. 1–4.

[4] Paul Benioff. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines". In: *Journal of Statistical Physics* 22.5 (May 1980), pp. 563–591. DOI: `10.1007/BF01011339`.

[5] Gérard Biau and Erwan Scornet. "A random forest guided tour". In: *TEST* 25.2 (June 2016), pp. 197–227. ISSN: 1863-8260. DOI: `10.1007/s11749-016-0481-7`. URL: `https://doi.org/10.1007/s11749-016-0481-7`.

[6] Trevor J. Bihl, Kenneth W. Bauer, and Michael A. Temple. "Feature Selection for RF Fingerprinting With Multiple Discriminant Analysis and Using ZigBee Device Emissions". In: *IEEE Transactions on Information Forensics and Security* 11.8 (2016), pp. 1862–1874. DOI: `10.1109/TIFS.2016.2561902`.

[7] Langley P. Blum A. "Selection of relevant features and examples in machine learning." In: *Artificial Intelligence* (1997). Cited By :1, p. 245. URL: `www.scopus.com`.

[8] Kelly Boothby et al. "Next-generation topology of d-wave quantum processors". In: *arXiv preprint arXiv:2003.00133* (2020).

[9] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. “A Training Algorithm for Optimal Margin Classifiers”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT ’92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[10] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2006), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[11] Evgeny Byvatov and Gisbert Schneider. “Support vector machine applications in bioinformatics.” In: *Applied bioinformatics* 2.2 (2003), pp. 67–77.

[12] Jun Cai, William G. Macready, and Aidan Roy. “A practical heuristic for finding graph minors”. In: (2014). arXiv: 1406.2741 [quant-ph].

[13] Anwesha Chakraborty et al. “Application of graph theory in social media”. In: *International Journal of Computer Sciences and Engineering* 6.10 (2018), pp. 722–729.

[14] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers Electrical Engineering* 40.1 (2014). 40th-year commemorative issue, pp. 16–28. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2013.11.024. URL: https://www.sciencedirect.com/science/article/pii/S0045790613003066.

[15] Vicky Choi. “Minor-embedding in adiabatic quantum computation: I. The parameter setting problem”. In: *Quantum Information Processing* 7.5 (Oct. 2008), pp. 193–209. ISSN: 1573-1332. DOI: 10.1007/s11128-008-0082-9. URL: https://doi.org/10.1007/s11128-008-0082-9.

[16] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018.

[17] Vasil S. Denchev et al. “What is the Computational Value of Finite-Range Tunneling?” In: *Physical Review X* 6.3 (Aug. 2016). ISSN: 2160-3308. DOI: 10.1103/physrevx.6.031015. URL: http://dx.doi.org/10.1103/PhysRevX.6.031015.

[18] R Dhanya et al. "F-test feature selection in Stacking ensemble model for breast cancer prediction". In: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (CoCoNet'19), pp. 1561–1570. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.167. URL: https://www.sciencedirect.com/science/article/pii/S1877050920311467.

[19] Artur J Ferreira and Mário AT Figueiredo. "Efficient feature selection filters for high-dimensional data". In: *Pattern Recognition Letters* 33.13 (2012), pp. 1794–1804.

[20] Richard P. Feynman. "Simulating Physics with Computers". In: *International Journal of Theoretical Physics* 21.6-7 (June 1982), pp. 467–488. DOI: 10.1007/BF02650179.

[21] Leslie R Foulds. *Graph theory applications.* Springer Science & Business Media, 2012.

[22] Yoav Freund, Robert Schapire, and Naoki Abe. "A short introduction to boosting". In: *Journal-Japanese Society For Artificial Intelligence* 14.771-780 (1999), p. 1612.

[23] Fred Glover. "Tabu Search—Part I". In: *ORSA Journal on Computing* 1.3 (1989), pp. 190–206. DOI: 10.1287/ijoc.1.3.190. eprint: https://doi.org/10.1287/ijoc.1.3.190. URL: https://doi.org/10.1287/ijoc.1.3.190.

[24] Fred W. Glover. "Tabu Search - Part II". In: *INFORMS J. Comput.* 2 (1990), pp. 4–32.

[25] Fred W. Glover and Gary A. Kochenberger. "A Tutorial on Formulating QUBO Models". In: *CoRR* abs/1811.11538 (2018). arXiv: 1811.11538. URL: http://arxiv.org/abs/1811.11538.

[26] Michel X. Goemans and David P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6 (Nov. 1995), pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: https://doi.org/10.1145/227683.227684.

[27] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[28] Andreas Grünauer and Markus Vincze. "Using Dimension Reduction to Improve the Classification of High-dimensional Data". In: *CoRR* abs/1505.06907 (2015). arXiv: `1505.06907`. URL: `http://arxiv.org/abs/1505.06907`.

[29] Laszlo Gyongyosi and Sandor Imre. "A Survey on quantum computing technology". In: *Computer Science Review* 31 (Feb. 2019), pp. 51–71. DOI: `10.1016/j.cosrev.2018.11.002`.

[30] Tin Kam Ho. "Random decision forests". In: 1 (1995), pp. 278–282.

[31] Thorsten Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features". In: *Machine Learning: ECML-98*. Ed. by Claire Nédellec and Céline Rouveirol. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142. ISBN: 978-3-540-69781-7.

[32] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: `10.1007/978-1-4684-2001-2_9`. URL: `https://doi.org/10.1007/978-1-4684-2001-2_9`.

[33] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. "Optimization by Simulated Annealing". In: *Science (New York, N.Y.)* 220 (June 1983), pp. 671–80. DOI: `10.1126/science.220.4598.671`.

[34] Huan Liu and R. Setiono. "Chi2: feature selection and discretization of numeric attributes". In: (1995), pp. 388–391. DOI: `10.1109/TAI.1995.479783`.

[35] Yuri Manin. "Computable and Noncomputable ("Vychislimoe i nevychislimoe"), Moscow: Sov". In: (1980).

[36] Oliver Mason and Mark Verwoerd. "Graph theory and networks in biology". In: *IET systems biology* 1.2 (2007), pp. 89–119.

[37] Catherine C McGeoch et al. "Practical annealing-based quantum computing". In: *Computer* 52.6 (2019), pp. 38–46.

[38] Rajdeep Kumar Nath, Himanshu Thapliyal, and Travis S. Humble. "Quantum Annealing for Automated Feature Selection in Stress Detection". In: (2021). arXiv: `2106.05134 [quant-ph]`.

[39] H. Neven et al. “QBoost: Large scale classifier training with adiabatic quantum optimization”. In: *Journal of Machine Learning Research* 25 (Jan. 2012), pp. 333–348.

[40] Hartmut Neven et al. “NIPS 2009 Demonstration: Binary Classification using Hardware Implementation of Quantum Annealing”. In: (Jan. 2010).

[41] Hartmut Neven et al. “Training a Binary Classifier with the Quantum Adiabatic Algorithm”. In: (2008). arXiv: `0811.0416 [quant-ph]`.

[42] Xuan Vinh Nguyen et al. “Effective Global Approaches for Mutual Information Based Feature Selection”. In: KDD ’14 (2014), pp. 512–521. DOI: `10.1145/2623330.2623611`. URL: `https://doi.org/10.1145/2623330.2623611`.

[43] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[44] Hanchuan Peng, Fuhui Long, and Chris Ding. “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy”. In: *IEEE Transactions on pattern analysis and machine intelligence* 27.8 (2005), pp. 1226–1238.

[45] Renfrey B Potts and Robert M Oliver. *Flows in transportation networks*. New York, Academic Press, 1972.

[46] Debbie L Hahs-Vaughn Richard G Lomax. *Statistical concepts: a second course*. Ed. by Routledge. 2013.

[47] Noelia Sánchez-Maroño et al. “Functional Networks and Analysis of Variance for Feature Selection”. In: (2006). Ed. by Emilio Corchado et al., pp. 1031–1038.

[48] Robert E. Schapire. “The Boosting Approach to Machine Learning: An Overview”. In: *Nonlinear Estimation and Classification*. Ed. by David D. Denison et al. New York, NY: Springer New York, 2003, pp. 149–171. ISBN: 978-0-387-21579-2. DOI: `10.1007/978-0-387-21579-2_9`. URL: `https://doi.org/10.1007/978-0-387-21579-2_9`.

[49] Ruslan Shaydulin et al. “A Hybrid Approach for Solving Optimization Problems on Small Quantum Computers”. In: *Computer* 52.6 (2019), pp. 18–26. DOI: `10.1109/MC.2019.2908942`.

[50] Chien-Chung Shen and Wen-Hsiang Tsai. “A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion”. In: *IEEE Transactions on Computers* 100.3 (1985), pp. 197–203.

[51] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[52] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.

[53] F.E.H. Tay and Lixiang Shen. "A modified Chi2 algorithm for discretization". In: *IEEE Transactions on Knowledge and Data Engineering* 14.3 (2002), pp. 666–670. DOI: 10.1109/TKDE.2002.1000349.

[54] K. Torkkola. "Feature extraction by non-parametric mutual information maximization". English. In: *Journal of Machine Learning Research* 3 (2003). cited By 510, pp. 1415–1438. ISSN: 15324435. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-1942450610&partnerID=40&md5=ba1814e755507a826d46fc737a0fb844.

[55] Zilin Zeng et al. "A novel feature selection method considering feature interaction". In: *Pattern Recognition* 48.8 (2015), pp. 2656–2666. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2015.02.025. URL: https://www.sciencedirect.com/science/article/pii/S0031320315000850.

[56] Zhihong Zhang and Edwin Hancock. "A Graph-Based Approach to Feature Selection". In: (May 2011). DOI: 10.1007/978-3-642-20844-7_21.