Benchmarking machine learning models for quantum state classification

Edoardo Pedicillo[1,2,*], *Andrea* Pasquale[1,2,**], and *Stefano* Carrazza[1,2,3,***]

[1]TIF Lab, Dipartimento di Fisica, Università degli Studi di Milano and INFN Sezione di Milano, Milan, Italy.
[2]Quantum Research Center, Technology Innovation Institute, Abu Dhabi, UAE.
[3]CERN, Theoretical Physics Department, CH-1211 Geneva 23, Switzerland.

Abstract. Quantum computing is a growing field where the information is processed by two-levels quantum states known as *qubits*. Current physical realizations of qubits require a careful calibration, composed by different experiments, due to noise and decoherence phenomena. Among the different characterization experiments, a crucial step is to develop a model to classify the measured state by discriminating the ground state from the excited state. In this proceedings we benchmark multiple classification techniques applied to real quantum devices.

1 Introduction

During the last decade, we have witnessed the spread of quantum devices granting users the possibility to start deploying quantum algorithms on quantum hardware. The typical way of accessing quantum processing units (QPUs) nowadays is through cloud providers. Although this approach is user-friendly, it tends to hide complications related to the deployment and maintenance of self-hosted QPUs, which are now accessible to research institutions.

In order to fill this gap, middleware open-source frameworks are fundamental to accelerate the growth and the development of research in quantum computing. An example of such frameworks is `Qibo` [1, 2], a full-stack quantum computing library providing an Application Programming Interface (API) to build quantum computing algorithms based on circuit and adiabatic paradigms. The latest addition to the Qibo framework are two modules dedicated to hardware control and characterization: `Qibolab` [3, 4] and `Qibocal` [5, 6].

Hardware control is not sufficient to deploy algorithms on quantum hardware. Current quantum technologies, especially superconducting devices, require a detailed characterization of the experimental setup, which is achieved by performing several experiments [7] aim at fine-tuning specific parameters. The need for calibration lies in the fact that superconducting qubits suffer from both noise and decoherence [8] phenomena which can lead to a shift in the characterization parameters. To address this issue there have been some works in order to automate the calibration process using direct acyclic graphs [9] and optimization techniques. In this context, several calibration libraries have appeared over the last years [9–11], including `Qibocal`.

*e-mail: edoardo.pedicillo@unimi.it
**e-mail: andrea.pasquale@unimi.it
***e-mail: stefano.carrazza@unimi.it

Among the different experiments required to achieve high-fidelity operations, we are going to focus on the protocol which trains a Machine Learning (ML) model to classify the measured state of a qubit. In the case of an ideal system, it should be quite straightforward to discriminate the state $|0\rangle$ from the state $|1\rangle$. However, this task can become non-trivial due to decoherence phenomena which tend to produce errors. This is why it is necessary to train a ML model to perform correctly the classification.

Such classification can be performed directly on the instrument which is collecting the measurements or can be executed afterwards as a post-processing operation. Although the first approach is generally faster, some devices have limited capabilities when it comes to storing and training ML models. This is why they may fall back on simplified models which can have worse performances. On the other hand, performing the classification outside the instrument can lead to better performances, but it may introduce some overhead.

In this proceedings we investigate this problem by benchmarking several ML classifiers trained on data obtained by real quantum devices.

2 Methods

2.1 Measuring a qubit

In this section we are going to briefly present how qubits are measured in the case of superconducting devices. The most known technology to realize superconducting qubits are transmons [12], weakly anharmonic oscillators made using Josephson junctions [13].

Transmons are measured by dispersively coupling them with resonators, also known as *dispersive readout*. It can be shown that in the non-resonant limit $|\Delta| \gg g$ the qubit-resonator system can be described by the following effective Hamiltonian [14, 15]:

$$H \approx \hbar \underbrace{\left(\omega_r + \frac{g^2}{\Delta}\sigma_z\right)}_{w} a^\dagger a + \frac{1}{2}\hbar\left(\omega_a + \frac{g^2}{\Delta}\right)\sigma_z, \tag{1}$$

where Δ is the detuning between the frequency of the resonator ω_r and the qubit frequency ω_a and g is the resonator-qubit coupling. The transition frequency of the resonator w is now affected by the state of the qubit:

$$w = \begin{cases} w_r + \frac{g^2}{\Delta}, & \text{if } |0\rangle \\ w_r - \frac{g^2}{\Delta}, & \text{if } |1\rangle \end{cases} . \tag{2}$$

We can detect this shift, and therefore identify the qubit state, by sending a readout probe signal generated using Arbitrary Waveform Generators (AWGs). We can extract the reflected amplitude and the phase of the microwave signal against the probe frequency ω. To properly observe the frequency shift we need to fine-tune the beam power to reach the dispersive-regime.

Another way to visualize the population of the ground and excited state is by demodulating the microwave probe into its in-phase I and quadrature component Q. By demodulating we are projecting the signal from the waveform space onto a 2D plane with coordinates I and Q. The ground and the excited states are identified by the two clouds shown in Fig. 1. This distribution is caused by decoherence and noise mostly coming from amplifiers [16].

We can expect that a generic measurement will generate some points in the IQ plane and we are interested in identifying if the point observed corresponds to $|0\rangle$ or $|1\rangle$. This problem is well known in ML and it is often referred to as *classification* or *discrimination* [17]. There are several models which can tackle this problem as we will show shortly.

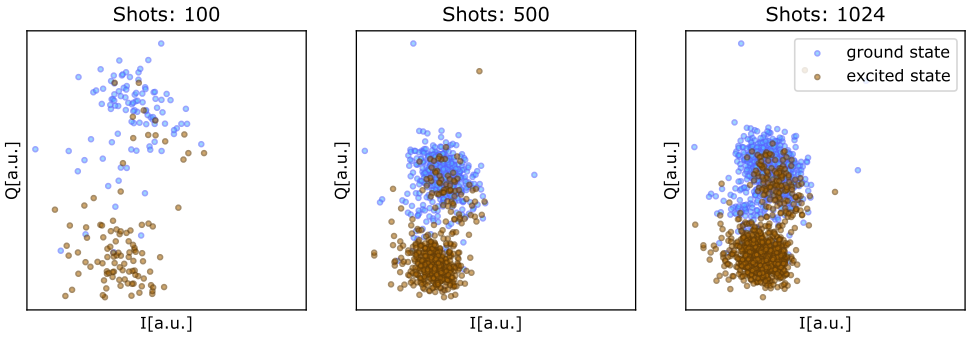

Figure 1: IQ components for different shots acquired by a real quantum device.

This classification experiment is usually performed after the calibration of the π-pulse, which aims at fine-tuning the parameters of a drive pulse to produce the excited state $|1\rangle$. The raw data are generated using two different sequences of pulses. We produce a series of 0s by simply measuring the state of the qubit, while the 1s are producing by measuring the qubit after sending a π-pulse. For each shot we store both the I and the Q component. As we can see in Fig. 1, the distribution becomes more complicated as the number of shots increases. We can also observe that some excited states decay into the ground state due to short lifetime of the qubit. Finally, the data are fed into a classification model which will be trained accordingly.

2.2 Classification models

It follows a brief description of the models that we have trained, including some of their strengths and weaknesses.

Linear SVM

A support vector machine (SVM) is a collection of supervised learning algorithms, that can be used both for classification and regression. Its versatility comes from the possibility of using different kernels, the most used is the linear kernel. As a classifier, the linear SVM is trained in order to find the best hyperplane that separates the two classes. The advantages of using this model are its robustness against overfitting due to its regularization parameter and its memory efficacy, since it uses a subset of training points, called *support vectors*, in the decision function. It is suitable when the number of dimensions exceeds the sample's one. On the other hand, it is sensitive to outliers that might affect the separation hyperplane, it is computationally expensive for large datasets and does not directly provide probability estimates.

AdaBoost

The main idea behind the AdaBoost algorithm is to generate and train a sequence of models, usually decision trees, that are slightly better than random guessing, usually called *weak learners*. Given a new input, the model classifies it by collecting the output of each weak learner and combining them with suitable weights. Using AdaBoost, it is less likely that

overfitting will occur since it uses an ensemble of weak learners that underfits the training set. The convergence is guaranteed, since the training error decreases exponentially with the number of epochs. However, it is sensitive to noisy data, which can negatively impact performance, and computationally more expensive than other algorithms. The training of the different learners cannot be parallelized, unlike random forests.

Gaussian Process

A Gaussian process classifier works by generating a function, *nuisance function*, $f(x)$ from a Gaussian process, and then squashing it through a function σ called link function (usually the logistic logit function). Therefore, the class probability is $\sigma(f(x))$. The values of f are not particularly relevant for the model, since the probability of belonging to a class is given by the average of class probabilities over the nuisance function space. Although the logistic link function integral cannot be calculated analytically, it can be easily approximated in the binary case. On the other side, it could be computationally demanding and may not scale efficiently to large datasets due to the lack of sparsity.

Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the *naive* assumption of conditional independence between every pair of features given the value of the class variable. It usually requires a small amount of training data, and the features' independence Ansatz could solve problems related to the curse of dimensionality. It could handle missing values and irrelevant features effectively. In the worst-case scenario, the independence assumption may be incorrect, or outliers and imbalanced datasets may reduce the model's effectiveness.

Neural Network

A neural network is a collection of connected nodes, each of which performs a specific operation on the input data. The connection topology and operation of the nodes differ between neural network models. We use a Feedforward Neural Network (FNN) in this proceedings, which is distinguished by one-way information transmission and the division of nodes into layers that are fully connected with the previous and next ones. When a non-linear approach is required, the FNN can be used for both regression and classification. Neural Networks have numerous advantages, including the ability to learn complex patterns and relationships in the training dataset, extreme flexibility, and good performance on large datasets with high dimensionality. The careful tuning of the hyperparameters to avoid overfitting and the training could require high computational costs. Because the loss function is non-convex, there can be several local minima, which suggests that the model is sensitive to weight initialization.

Random Forest

The Random Forest is a collection of trees that are generated by selecting a random sample of the training set with replacement. The best split of each tree could be evaluated either from all features or a random subset. This bootstrapping technique reduces model variance and the risk of overfitting, but it may slightly increase bias. The model predicts the class by averaging the output of each tree, which cancels out some errors. Although it is possible to parallelize tree construction and predictions, it may be computationally expensive for large datasets during training.

Radial Basis Function Support Vector Machine

Support Vector Machine with radial basis function [1] as kernel. It is effective for non-linearly separable data through the use of kernel functions, and it has only two hyperparameters that require careful selection and tuning.

Fidelity fit

This method [18] evaluates the axis which passes through the centroids of the two clusters in the training set, and determines the optimal threshold that maximizes the difference between the cumulative distributions of the projections along the axis. Predicting the state of a new point involves determining its relative position to the threshold based on its projection along the axis. Due to its small complexity, it has high interpretability, but it does not evaluate the probabilities.

2.3 Training

The initial step involved splitting the dataset into a test set, containing 25% of the data, and a training set. Following this, we optimize the hyperparameters for each classifier whenever possible.

The neural network was initialized by the `Keras` [19] library, its hyperparameters optimization was carried out by a variant of the `HyperBand` [20] algorithm provided by `Keras Tuner`, the fidelity fit model was implemented from scratch in `Qibocal`, and the other models are those available in the `scikit-learn` [21] library, their best hyperparameters are found using a grid search algorithm. The hyperparameter optimization details are tabulated in Tab. 1.

3 Results

A visualization of the classifiers' prediction capability is shown in Fig. 2. The scattered points represent the testing sample while we show in the background the prediction for each model. All models seem to have comparable performances as shown in Tab. 2 with accuracies distributed in a narrow interval around 0.90. Looking at the ROC curves in Fig. 3 we can conclude that the performances between all the models are similar with a slight advantage of the Neural Network that presents the best AUC. Given that all the models perform similarly in terms of all the quantities that describe the quality of a classifier (accuracy, true positive rate, false positive rate, AUC), selecting the optimal one is not trivial.

All the classifiers have testing times that are reasonably fast, ranging from 0.07 to 132.93 micro-seconds. These significant variations in testing time suggest that choosing the wrong classifier may generate an overhead which scale with the number of shots. This could affect several quantum algorithm such as variational quantum circuits where the training requires generating thousands of shots in each epoch. Also, the length of training period differs substantially. In comparison to the other classifiers, Gaussian Process and Neural Network have much longer training periods. On the other hand, Naive Bayes has the quickest training time at 0.005 seconds.

[1] Given $\mathbf{x}$ and $\mathbf{x}'$ as two feature vectors in the input space the radial basis function is defined as

$$K(\mathbf{x}, \mathbf{x}') = exp(-\gamma \left\| \mathbf{x} - \mathbf{x}' \right\|^2).$$

Model	Hyperparameters	Configuration space
Ada Boost	estimator number	[10, 200]
	learning rate	[0.1, 1]
	algorithm	*SAMME*, *SAMME.R*
Neural Network	size first layer	16, 1056
	size second layer	16, 1056
	activation function	*relu*, *sigmoid*, *tanh*, *RBF*
	learning rate	10^{-4}, 10^{-2}
	optimizers	*Adam*, *Adagrad*, *SGD*, *RMSprop*
Random Forest	estimators number	[10, 200]
	criterion	*gini*, *entropy*, *log_loss*
	number of feature for splitting	*sqrt*, *log2*, *all*
RBF SVM	regularization parameter	[0.01, 2]
	degree	2, 3, 4

Table 1: Table listing all the hyperparameters that were optimized and their corresponding values.

Name	Accuracy	Test Time (μs)	Train Time (s)	True Positive Rate	False Positive Rate
Ada Boost	0.904	16.22	0.401	0.94	0.13
Linear SVM	0.905	13.13	0.40	0.94	0.12
Gaussian Process	0.904	78.97	319.728	0.93	0.12
Naive Bayes	0.901	0.57	0.005	0.92	0.12
Fidelity Fit	0.904	0.07	0.129	0.95	0.14
Random Forest	0.890	24.47	0.720	0.91	0.12
RBF SVM	0.902	54.29	0.163	0.94	0.13
Neural Network	0.903	132.93	59.793	0.95	0.15

Table 2: Table collecting the different parameters that we took in consideration to benchmark the different models.

In conclusion, Fidelity Fit is the best option if testing time is an important consideration because of its greatly reduced training time and very good accuracy. However, Linear SVM, Ada Boost, or Naive Bayes can be taken into consideration since they offer similar accuracies, and they still have rather quick testing times. It is crucial to remember that the optimal classifier can change depending on the dataset and particular issue you are attempting to solve. To ensure the reliability of the results, it is advised to further analyze the classifiers using cross-validation methods and study the correlation between their performances and the quality of the qubits.

4 Outlook

In this proceedings we have considered a specific characterization experiment which is performed when calibrating QPUs: the construction of a model to perform the classification between the ground state and the excited state. We have compared the performance of several classifiers which have been trained on data acquired by real quantum processors at the

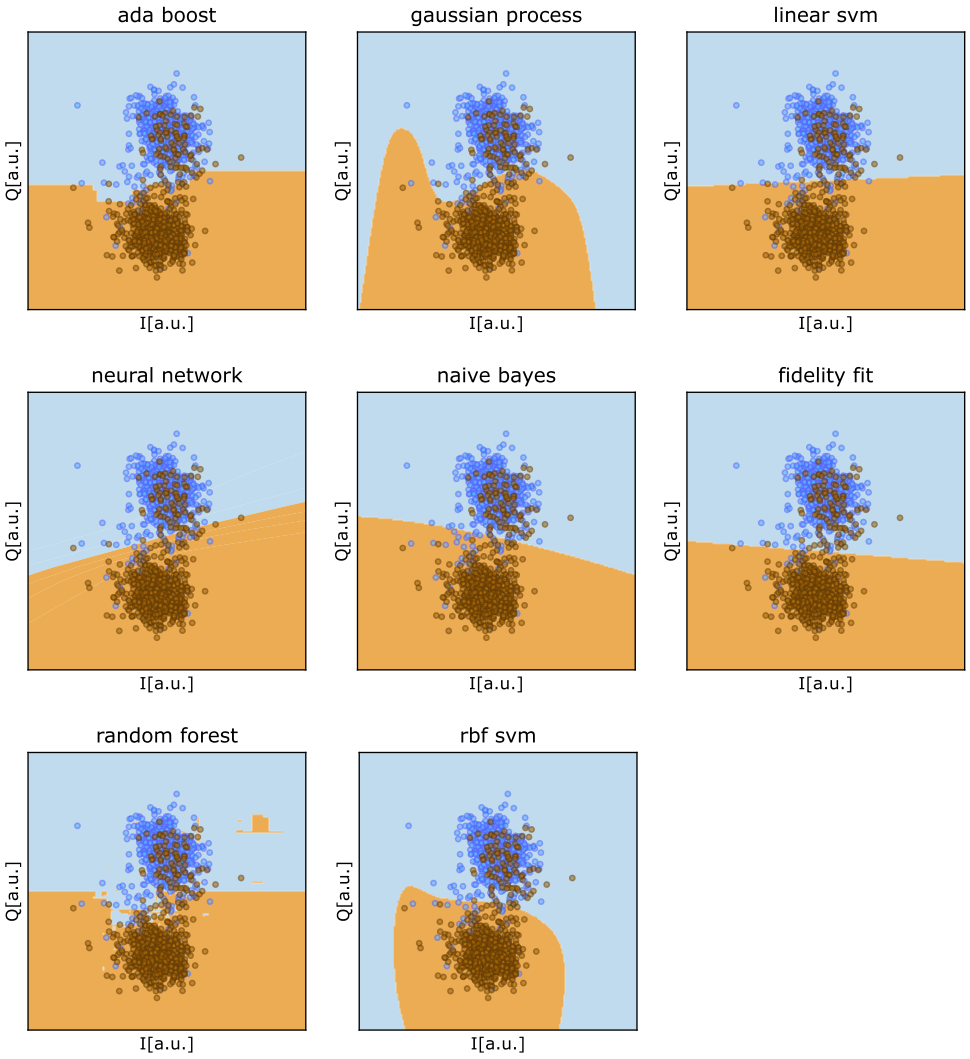

Figure 2: Classifiers' predictions (background colors) with test set (scattered points).

Quantum Research Center of Technology Innovation Institute (TII) in Abu Dhabi. The different models show a comparable performance when it comes to accuracy, testing time and training time. However, due to its simplicity we believe that the Fidelity Fit seems to be the best model for this particular dataset.

For future developments it would be interesting to perform the same study for three levels quantum system, *qutrits*. This could play a significant role in increasing the fidelity of two-qubit gates. Another interesting topic would be to see how the performance of the different models changes by looking at different qubits on the same chip, which will be particularly useful to perform noise aware classification.

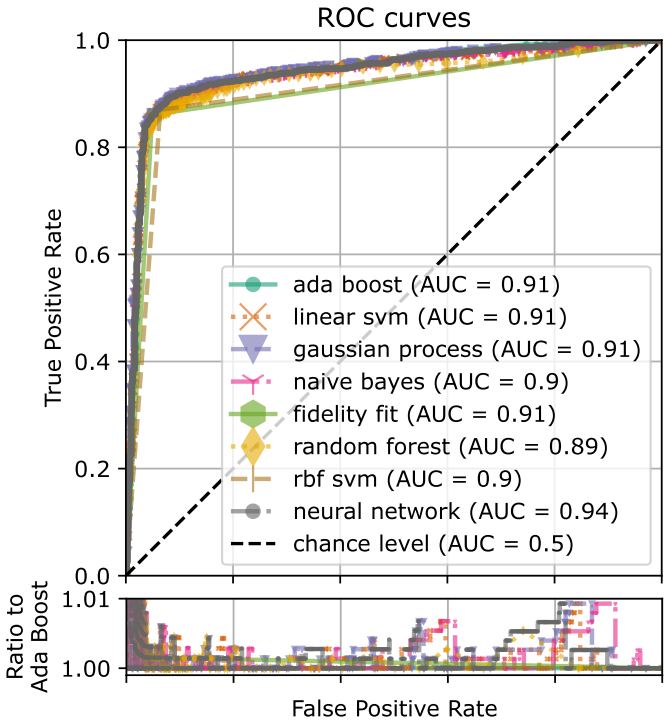

Figure 3: ROC curves computed for each classifiers (*top*) and ratio w.r.t. Ada Boost classifier (*bottom*).

5 Acknowledgements

The authors thank TII's Quantum Research Center for its support and Alessandro Candido for his help implementing the benchmarking.

References

[1] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J.I. Latorre, S. Carrazza, Quantum Science and Technology **7**, 015018 (2021)

[2] S. Efthymiou, S. Carrazza, R. Mello, Edoardo-Pedicillo, A. Pasquale, A. Sopena, shang-tai, M. Robbiati, C. Bravo-Prieto, AdrianPerezSalinas et al., *qiboteam/qibo: Qibo 0.1.14* (2023), `https://doi.org/10.5281/zenodo.7992830`

[3] S. Efthymiou, A. Orgaz-Fuertes, R. Carobene, J. Cereijo, A. Pasquale, S. Ramos-Calderer, S. Bordoni, D. Fuentes-Ruiz, A. Candido, E. Pedicillo et al., *Qibolab: an open-source hybrid quantum operating system* (2023), `2308.06313`

[4] S. Efthymiou, aorgazf, S. Carrazza, A. Pasquale, J. Cereijo, R. Carobene, Edoardo-Pedicillo, DavidSarlle, Simone-Bordoni, maxhant et al., *qiboteam/qibolab: Qibolab 0.0.4* (2023), `https://doi.org/10.5281/zenodo.7973899`

[5] A. Pasquale, S. Efthymiou, S. Ramos-Calderer, J. Wilkens, I. Roth, S. Carrazza, *Towards an open-source framework to perform quantum calibration and characterization* (2023), 2303.10397

[6] A. Pasquale, Edoardo-Pedicillo, DavidSarlle, S. Efthymiou, S. Carrazza, aorgazf, A. Sopena, maxhant, A. Candido, M. Robbiati et al., *qiboteam/qibocal: Qibocal 0.0.2* (2023), https://doi.org/10.5281/zenodo.7957542

[7] Y.Y. Gao, M.A. Rol, S. Touzard, C. Wang, PRX Quantum **2**, 040202 (2021)

[8] M. Schlosshauer, Physics Reports **831**, 1 (2019)

[9] J. Kelly, P. O'Malley, M. Neeley, H. Neven, J.M. Martinis, *Physical qubit calibration on a directed acyclic graph* (2018), 1803.03226

[10] N. Kanazawa, D.J. Egger, Y. Ben-Haim, H. Zhang, W.E. Shanks, G. Aleksandrowicz, C.J. Wood, Journal of Open Source Software **8**, 5329 (2023)

[11] K. Gulshen, J. Combes, M.P. Harrigan, P.J. Karalekas, M.P. da Silva, M.S. Alam, A. Brown, S. Caldwell, L. Capelluto, G. Crooks et al., *Forest Benchmarking: QCVV using PyQuil* (2019), https://doi.org/10.5281/zenodo.3455847

[12] J. Koch, T.M. Yu, J. Gambetta, A.A. Houck, D.I. Schuster, J. Majer, A. Blais, M.H. Devoret, S.M. Girvin, R.J. Schoelkopf, Phys. Rev. A **76**, 042319 (2007)

[13] B.D. Josephson, Phys. Lett. **1**, 251 (1962)

[14] A. Blais, R.S. Huang, A. Wallraff, S.M. Girvin, R.J. Schoelkopf, Phys. Rev. A **69**, 062320 (2004)

[15] A. Wallraff, D. Schuster, A. Blais, L. Frunzio, R. Huang, J. Majer, S. Kumar, S. Girvin, R. Schoelkopf, Nature **431**, 162 (2004)

[16] F. Arute, K. Arya, R. Babbush, D. Bacon, J.C. Bardin, R. Barends, R. Biswas, S. Boixo, F.G.S.L. Brandao, D.A. Buell et al., Nature **574**, 505 (2019)

[17] G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning* (Springer, 2013)

[18] M.D. Reed, L. DiCarlo, B.R. Johnson, L. Sun, D.I. Schuster, L. Frunzio, R.J. Schoelkopf, Physical Review Letters **105** (2010)

[19] F. Chollet et al., *Keras*, https://keras.io (2015)

[20] K. Team, https://keras.io/api/keras_tuner/tuners/hyperband/

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., Journal of Machine Learning Research **12**, 2825 (2011)