Text Classification with Born's Rule

Emanuele Guidotti
Institute of Financial Analysis
University of Neuchâtel, Switzerland
emanuele.guidotti@unine.ch

Alfio Ferrara
Department of Computer Science and Data Science Research Center
University of Milan, Italy
alfio.ferrara@unimi.it

Abstract

This paper presents a text classification algorithm inspired by the notion of superposition of states in quantum physics. By regarding text as a superposition of words, we derive the wave function of a document and we compute the transition probability of the document to a target class according to Born's rule. Two complementary implementations are presented. In the first one, wave functions are calculated explicitly. The second implementation embeds the classifier in a neural network architecture. Through analysis of three benchmark datasets, we illustrate several aspects of the proposed method, such as classification performance, explainability, and computational efficiency. These ideas are also applicable to non-textual data.

1 Introduction

In quantum physics, the superposition principle is the idea that a system can be in multiple states at the same time (e.g., a cat is simultaneously alive and dead in Schrödinger's thought experiment [22]). Here we show that text can be formally treated as a quantum system in a superposition of words. Our superposition-of-words model is a bag-of-words model where each word is treated as a quantum state. By exploiting this representation, we develop a supervised classifier based on key postulates of quantum mechanics, namely the Born rule [4].

The Born rule provides a link between the mathematical formalism of quantum theory and experiment, and as such is almost single-handedly responsible for practically all predictions of quantum physics.[1] In this paper, we represent documents and classes as quantum objects and we compute the probability of a document to collapse in a target class by applying the Born rule.

The paper is structured as follows. Section 2 presents the motivation of this work. Section 3 introduces the notation and some preliminary notions in quantum mechanics. Section 4 develops our classification algorithm. Section 5 embeds the classifier in a neural network architecture. Section 6 presents our empirical results. Finally, Section 7 gives our concluding remarks and discusses extensions to semantics and non-textual data.

To simplify extensions to this work, we implement our classification algorithm in scikit-learn [17] and we embed the classifier in a neural network architecture using pytorch [16]. All the code is available at `https://github.com/eguidotti/bornrule`.

[1] We refer the reader to [14] for an introduction to the Born rule.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

2 Motivation

In its most general form, the no-free-lunch theorem [23] implies that only prior knowledge makes it possible to generalize from the training examples to novel test examples. As quantum mechanics represents our understanding of nature at the deepest level [13], we wonder whether quantum-inspired machine learning may incorporate a fundamental form of prior knowledge. Our hope is that such knowledge results in more efficient algorithms, with better generalization ability and shorter computational times. Such algorithms would be also beneficial in terms of explainability and interpretability [8], as their interpretation is immediately inherited from the physical model [18].[2]

This work should not be confused with quantum computing, which harnesses the properties of quantum states to perform calculations [1], or with quantum machine learning [3], which explores how to devise and implement quantum software that could enable machine learning that is faster than that of classical computers [6]. We refer to our work as quantum-inspired machine learning, in that we use quantum theory to derive machine learning algorithms that can run on classical computers. Although this is not the first time quantum theory is applied to machine learning (see e.g., [15]), this is, to our knowledge, the first step to develop a new general-purpose algorithm for supervised classification based on key postulates of quantum mechanics, namely the Born rule.

Here we focus on text classification, as the analogy with quantum systems is straightforward.[3] However, the formalism we present is general and the implementations we derive can be used in practice as general-purpose classifiers.

3 Notation

Let $\mathbf{x}$ be a feature vector with elements x_j for $j = 1, \ldots, J$. Let $\mathbf{y}$ be a probability vector with elements $y_k \geq 0$ for $k = 1, \ldots, K$, and such that $\sum_k y_k = 1$. Our goal is to learn a function g such that $\mathbf{y} = g(\mathbf{x})$. Then, given a test instance $\mathbf{x}^0$, we predict the probabilities $\mathbf{y}^0 = g(\mathbf{x}^0)$ and select the class $k^* = \operatorname{argmax}_k y^0_k$ for classification.

We start by giving some preliminary notions in quantum mechanics.

Wave function. In quantum physics, a system is regarded as a superposition of states $|s\rangle$ and, using Dirac's notation [7], it is represented by a wave function $|\psi\rangle$:

$$|\psi\rangle = \sum_s \psi_s |s\rangle \quad \text{with} \quad \psi_s \in \mathbb{C}. \tag{1}$$

Born rule. In the Copenhagen interpretation, the modulus squared of the inner product is interpreted as the (unnormalized) probability of the wave function $|\psi\rangle$ collapsing to a new wave function $|\psi'\rangle$:

$$P(\psi \to \psi') = |\langle\psi'|\psi\rangle|^2 = \left|\sum_s \psi'_s \psi_s\right|^2; \tag{2}$$

where ψ'_s denotes the complex conjugate of ψ'_s. This is known as the Born rule, and it is one of the fundamental postulates of quantum mechanics.

Wave coefficients. From (1) and (2), we notice that the coefficient $|\psi_s|^2$ represents the (unnormalized) probability of the wave function $|\psi\rangle$ to collapse in state $|s\rangle$:

$$P(\psi \to s) = |\langle s|\psi\rangle|^2 = |\psi_s|^2. \tag{3}$$

4 Classification Algorithm

Let the feature vector $\mathbf{x}$ contain only non-negative elements such that $x_j \geq 0$ for all j (e.g., word counts or tf-idf weights). We regard x_j as the (unnormalized) probability of the data instance (e.g., document) to collapse in the j-th feature (e.g., word). We represent the j-th feature as a quantum state

[2] We refer the reader to [19] for an introduction to quantum mechanics and its interpretation.

[3] We refer the reader to [2, 5, 21, 27, 28] for quantum language models.

$|j\rangle$, and we represent the data instance with a superposition of states $|x\rangle = \sum_j \alpha_j |j\rangle$. According to (3) we have $x_j = P(x \to j) = |\alpha_j|^2$, which implies the natural choice $\alpha_j = \sqrt{x_j}$.

$$|x\rangle = \sum_j \alpha_j |j\rangle = \sum_j \sqrt{x_j}\, |j\rangle \tag{4}$$

In a similar way, we represent the k-th class with a wave function $|\psi^{(k)}\rangle = \sum_j \psi_j^{(k)} |j\rangle$, and we obtain the coefficients $\psi_j^{(k)}$ by setting the transition probability from $|\psi^{(k)}\rangle$ to $|j\rangle$ equal to the conditional probability of feature j given class k, which we write as $P_{j|k}$. According to (3) we have $P_{j|k} = P(\psi^{(k)} \to j) = |\psi_j^{(k)}|^2$, which implies the natural choice $\psi_j^{(k)} = \sqrt{P_{j|k}}$.

$$|\psi^{(k)}\rangle = \sum_j \psi_j^{(k)} |j\rangle = \sum_j \sqrt{P_{j|k}}\, |j\rangle \tag{5}$$

Finally, we obtain the classification probability y_k by computing the probability of $|x\rangle$ to collapse in $|\psi^{(k)}\rangle$. By substituting (4) and (5) in (2), the unnormalized probabilities are:

$$u_k = P(x \to \psi^{(k)}) = |\langle\psi^{(k)}|x\rangle|^2 = \left|\sum_j \psi_j^{(k)} \alpha_j\right|^2 = \left|\sum_j \sqrt{P_{j|k}\, x_j}\right|^2; \tag{6}$$

and the normalized probabilities are $y_k = u_k / \sum_k u_k$.

4.1 Training

To obtain the conditional probability $P_{j|k}$ in (6) we proceed as follows. Given a training set $\{(x^{(n)}, y^{(n)})\}_{n=1,\dots,N}$, we normalize each feature vector $x^{(n)}$ such that it sums up to 1:

$$z_j^{(n)} = \frac{x_j^{(n)}}{\sum_{j'} x_{j'}^{(n)}}. \tag{7}$$

Then, we compute the conditional probability $P_{j|k}$ from the (unnormalized) joint probability P_{jk}:

$$P_{jk} = \sum_n z_j^{(n)} y_k^{(n)}; \quad P_{j|k} = \frac{P_{jk}}{\sum_{j'} P_{j'k}}. \tag{8}$$

4.2 Regularization

We observe that if $P_{j|k}$ is constant for $k = 1, \dots, K$, then the j-th addend increases the summation in (6) by the same value for all classes. As $\sqrt{\cdot}$ is a monotonic transformation of the summation, the j-th addend does not alter the ranking of the probabilities u_k, thus being irrelevant for the final classification $k^* = \arg\max_k y_k = \arg\max_k u_k$. To regularize the predictions, we shrink the contribution of irrelevant addends towards zero by re-weighting the summation in (6).

Let us rewrite, for ease of notation, the probabilities $P_{j|k}$ as some weights $W_{jk} \geq 0$. Then, j is irrelevant if W_{jk} is constant for all k. We normalize W_{jk} such that the weights of the classes sum up to 1 for each j, that is:

$$W_{k|j} = \frac{W_{jk}}{\sum_{k'} W_{jk'}}. \tag{9}$$

An irrelevant j maximizes the entropy $H_j = \sum_k W_{k|j} \ln(W_{k|j})$, as $W_{k|j}$ is uniformly distributed across the classes k. Thus, we introduce the following weights that range between 0 (irrelevant with maximum entropy) and 1 (relevant with null entropy):

$$H_j = 1 - \frac{H_j}{H_{max}} = 1 + \frac{\sum_k W_{k|j} \ln(W_{k|j})}{\ln(\sum_k 1)}. \tag{10}$$

Finally, we use H_j in (10) to re-weight the summation in (6), which becomes:

$$u_k = \sum_j H_j \sqrt{P_{j|k}\, x_j}^{\,2}. \tag{11}$$

4.3 Generalization

To simplify ablation studies, we generalize (11) as follows:

$$u_k = \sum_j H_j^h W_{jk}^a x_j^a \quad \text{with} \quad W_{jk} = \frac{P_{jk}}{(\sum_{j'} P_{j'k})^b (\sum_{k'} P_{jk'})^{1-b}}; \tag{12}$$

where H_j is given in (9)–(10), P_{jk} is given in (8), and $a > 0$, $b \geq 0$, and $h \geq 0$ are the model hyper-parameters. Here, we are mainly interested in the choice $a = \frac{1}{2}$, $b = 1$, and $h = 1$, which corresponds to the original model (11). Another special configuration is $a = 1$, $b = 0$, and $h = 0$, which corresponds to $u_k = \sum_j P_{k|j} x_j$ where $P_{k|j}$ is the conditional probability of k given j. This configuration offers a natural benchmark for our quantum approach in that it computes u_k according to classical probability theory.

4.4 Explainability

The contribution of the j-th feature to the total probability u_k (local explanation) is given by the addend $H_j^h W_{jk}^a x_j^a$ in (12). Therefore, the most influential feature for the classification k is given by $j = \operatorname{argmax}_j H_j^h W_{jk}^a x_j^a$. In general, we use $H_j^h W_{jk}^a x_j^a$ to rank the features by the degree in which they contribute to the classification k.

The explanation at the class level (global explanation) is obtained by investigating the product $H_j^h W_{jk}^a$ in (12), regardless of the vector x. The global most influential feature for each class k is given by $j_k = \operatorname{argmax}_j H_j^h W_{jk}^a$. In general, we use $H_j^h W_{jk}^a$ to rank the features by their global importance with respect to class k.

4.5 Computational Complexity

In the training phase, the algorithm (11), and more generally (12), learns the joint probability P_{jk} in (8) by multiplying the $J \times N$ matrix of elements $x_j^{(n)}$ with the $N \times K$ matrix of elements $y_k^{(n)}$. Entry P_{jk} is given by the inner product of the j-th row of the left matrix (which has N entries) and the k-th column of the right matrix (which has N entries), so computing it takes time $O(N)$. We do this once per element. Since the output matrix has dimension $J \times K$, there are $O(JK)$ elements to consider and the total work is done in $O(NJK)$. That is, the training time is at most linear in the number of samples (N), in the number of features (J), and in the number of classes (K).

In the prediction phase, we compute u_k (and y_k) in (11), and more generally (12). As these are all elementwise operations on the $J \times K$ matrix of elements P_{jk}, the total work is done in time $O(JK)$. That is, the prediction time is at most linear in the number of features (J), and in the number of classes (K), and it does not depend on the number of training samples (N).

Finally, we notice that the computational complexity can be further improved by using sparse matrices, and all the operations involved in the training and prediction phases can be easily parallelized (e.g., on GPUs). Thus, we expect the method to be highly scalable.

5 Neural Architecture

A major limitation of the algorithm presented in Section 4 is that it can be applied only when $x_j \geq 0$. Here we embed the method in a more flexible architecture that admits $x_j \in \mathbb{C}$.

Let us assume that a data instance (e.g., document) can be represented as a superposition of some hidden states $|s\rangle$ for $s = 1, \ldots, S$ (e.g., word embeddings). Then, we write its wave function $|\psi\rangle = \sum_s \alpha_s |s\rangle$ where the coefficients α_s generally depend on the feature vector x (e.g., words). We represent such coefficients with a neural network $\alpha_k = \alpha_s(x)$ that maps the feature vector $x \in \mathbb{C}^J$ to the vector of wave coefficients $\alpha \in \mathbb{C}^S$. Then, we write the wave function of class k as $|\psi^{(k)}\rangle = \sum_s \psi_s^{(k)} |s\rangle$ where the coefficients $\psi_s^{(k)}$ depend on k and s, but not on the feature vector x. Finally, we use the Born rule in (2) to compute the probability of $|\psi\rangle$ to collapse into $|\psi^{(k)}\rangle$:

$$u_k = P(\psi \to \psi^{(k)}) = |\langle \psi^{(k)} | \psi \rangle|^2 = \left| \sum_s \psi_s^{(k)} \alpha_s(x) \right|^2. \tag{13}$$

Equation (13) is read as a neural network $u_k = \Phi(v)$ where Φ is a matrix of elements $\Phi_{ks} = \psi_s^{(k)}$, $v = \Psi(x)$ is the output of the previous layer, Φv denotes the matrix product $\sum_s \Phi_{ks} v_s$, and the activation function $\Phi(\cdot) = |\cdot|^2$ is the modulus squared. Finally, we apply a normalization layer to obtain the probabilities $y_k = u_k / \sum_k u_k \in [0, 1]$. An illustration is given in Figure 1.

Figure 1: Born Layer (BL) architecture.

Our method is similar to the usual approach where the first part of a network is assumed to learn log-probabilities, and the final classification layer converts them into probabilities by applying the softmax function. Here, the first part of the network is assumed to learn the coefficients of a wave function, and the final classification layer converts them into probabilities by applying the Born rule.

5.1 Initialization

We initialize the weights Φ_{ks} such that the wave function $|\psi^{(k)}\rangle$ has an equal probability to collapse in any state $|s\rangle$. To this end, let us write the complex-valued weights $\Phi_{ks} = \rho_{ks} e^{i\theta_{ks}}$, where i is the imaginary unit, $\rho_{ks} \geq 0$, and $\theta_{ks} \in [0, 2\pi)$. We recall from (3) that the probability of $|\psi^{(k)}\rangle$ to collapse in $|s\rangle$ is $P(\psi^{(k)} \to s) = |\langle\psi_s^{(k)}|\rangle|^2 = |\Phi_{ks}|^2 = \rho_{ks}^2$. We set $\rho_{ks}^2 = \rho^2$ so that the (unnormalized) probability is constant for all k and s. To normalize the probability, we set $S\rho^2 = 1$, where S is the number of states (input dimension of the layer). We obtain $\rho = 1/\sqrt{S}$, which resembles the initialization in [9] and [10]. Finally, we sample θ_{ks} from a uniform distribution in the interval $[0, 2\pi)$, such that the weights Φ_{ks} are uniformly distributed in the complex circle (isotropy).

$$\Phi_{ks} = \frac{e^{i\theta_{ks}}}{\sqrt{S}} \quad \text{with} \quad \theta_{ks} \sim U(0, 2\pi): \tag{14}$$

When the feature vector x is a (unnormalized) probability vector as in Section 4, the (11) can be written as the neural network in (13) where we use $S = J$ and $\Psi_s(x) = \sqrt{x_s}$. In this case, it is interesting to initialize the weights in (13) with the corresponding weights developed in (11), that is $\Phi_{ks} = H_s \sqrt{P_{sjk}}$.[4]

5.2 Explainability

We notice that the probabilities y_k are invariant under scaling and rotation of the coefficients $\psi_s^{(k)}$ in (13). To show that, we multiply $\psi_s^{(k)}$ by a scaling factor λ and a phase factor $e^{i\phi}$. Then, we substitute $\psi_s^{(k)} \to \lambda e^{i\phi} \psi_s^{(k)}$ in (13). The phase factor $e^{i\phi}$ vanishes when computing the modulus and the scaling factor λ vanishes when the probabilities are normalized.

The invariance by scaling and rotation implies that the weights Φ_{ks} have no absolute meaning, and they become meaningful only in relation with each other. To inspect the relations among states for a given class k, it is interesting to visualize the weights Φ_{ks} in the complex plane (with unlabelled axes). Here, the length of the vectors can be used to rank the states by importance, while the direction of the vectors produce constructive or destructive interference among the states (see Figure 5 for an illustration). For local explanations, $\Phi_{ks} v_s$ is used instead of Φ_{ks}.

[4] More precisely, we scale the weights by dividing them by their mean and by the square root of the number of features to mimic (14). This does not alter the model, which is invariant under scaling (see Section 5.2), but it helps to prevent vanishing or exploding gradients during backpropagation.

6 Empirical Results

We illustrate several aspects of our classifier using three well-established text classification benchmarks: 20Newsgroup[5], and the R8 and R52 subsets of Reuters 21578[6]. We perform tokenization using the function nltk.word_tokenize[7] and vectorize the text with TfidfVectorizer[8]. No other text transformation or cleaning procedure is performed. The final datasets are composed by: 20Newsgroup: 20 classes, 204 817 words, 11 314 training documents, and 7 532 test documents; R8: 8 classes, 33 593 words, 5 485 training documents, 2 189 test documents; R52: 52 classes, 38 132 words, 6 532 training documents, and 2 568 test documents.

All the results are obtained using Python 3.9 on a Google Cloud Virtual Machine equipped with CentOS 7, 12 vCPU Intel Cascade Lake 85 GB RAM, 1 GPU NVIDIA Tesla A100, and CUDA 11.5.

6.1 Training time, prediction time, and accuracy score

We compare our methodology against a baseline of six classifiers on the 20Newsgroup dataset. With Born Classifier (BC), we refer to the algorithm presented in Section 4, where weights are computed as in equation (8) and classification probabilities are calculated as in equation (11). The baseline is composed by the algorithms Decision Tree (DT), K-Nearest Neighbors (KNN), Random Forest (RF), Support Vector Machine (SVM), Multinomial Naive Bayes (MNB), and Logistic Regression (LR). For all the algorithms in the baseline, we use the corresponding implementation in scikit-learn. All the classifiers are executed on CPU with default parameters.

Figure 2: From left to right: training time, prediction time, and accuracy score on the 20Newsgroup dataset (y-axis) for several classifiers, in function of the fraction of data used for training (x-axis).

The comparison between BC and the baseline is reported in Figure 2, where we show the training time, prediction time, and accuracy score in function of the fraction of data used for training in 10 independent executions. Figure 2 shows that BC is fast to train, is fast to predict, can be accelerated on GPU, and it achieves the highest accuracy regardless of the size of the dataset. Moreover, as the amount of training data decreases, the accuracy gap between BC and the other classifiers widens.

6.2 Imbalanced data

While 20Newsgroup is almost balanced in terms of documents per class, R8 and R52 are not. In particular, the most frequent class in R52 contains 2 840 training samples, while the least frequent class contains only 1 document in the training set. Figure 3 reports the F1-macro score of BC and the baseline on the three datasets, which are increasingly imbalanced. BC outperforms the baseline models and the performance gap widens for more imbalanced data. The native capability of BC to work with imbalanced data can be traced back to (8), which computes the conditional probability of the features given the classes. Dividing the joint probability by the marginal effectively normalizes by the class imbalance.

[5] http://qwone.com/~jason/20Newsgroups
[6] http://archive.ics.uci.edu/ml/machine-learning-databases/reuters21578-mld
[7] See https://www.nltk.org/book/
[8] See https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html