POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

MTH6312 - Project

Variational autoencoder as a justification for naive Bayes, linear and quadratic discriminant analysis

Students :
Quentin Fournier #1865777
Charafeddine Talal #1968017

December 21, 2018

Contents

1 Introduction

Classical machine learning methods such as naive Bayes, linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA) have been applied with success to many different problems. However, all the above methods make assumptions on the data distribution. Although those methods tend to work well even when their assumptions are not met, one could look for a way to systematically justify their use.

As part of our project, we propose to learn a projection of the data that will verify all the assumptions made by naive Bayes, LDA and QDA. In order to do so, we used an unsupervised probabilistic neural network called a variational autoencoder. Such a model can learn a projection which tends to follow a normal distribution $\mathcal{N}(0, I)$. This allows us to evaluate the impact of violating – or respecting – the assumptions made by the three classifiers.

When applied on a real data set of credit card fraud detection, we observed a significant improvement for QDA and naive Bayes. More specifically, for a small trade-off in precision, the recall rate of both methods increase 7 fold. However, LDA performs only slightly better on the learn projection than on the original space.

Note that this project is open source[1] and that the code is available at this address: `https://github.com/qfournier/vae_justification`.

The rest of this work is organized as follows: section 2 describes in detail the methods used. Section 3 presents the experimental framework, the data set and the results. Finally, section 4 summarized this work.

[1]This project is under the MIT licence.

2 Methods

2.1 Variational Autoencoder

First, let us introduce the framework of an autoencoder (Hinton and Salakhutdinov, 2006). Given an input x, its projection z and its reconstruction x', an autoencoder is composed of two symmetric networks (Figure 1):

– Encoder: defined by the function $f(x) = z$ such that x is the input and z is the output of the network.

– Decoder: defined by the function $g(z) = x'$ such that z is the input and x' is the output of the network.

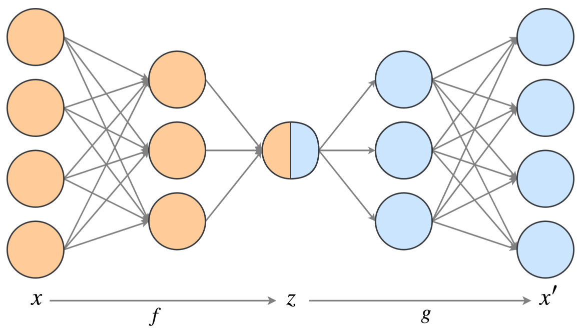


Figure 1: Framework of an autoencoder. Both the encoder and the decoder have one hidden layer.

The training objective is to minimize the distance between the input x and its reconstruction $g(f(x)) = x'$. It is necessary to limit the capacity of the model to copy its input on its output in order to force the autoencoder to extract useful properties. One can impose a regularization term or limit the dimension of the projection z.

A *variational autoencoder* (VAE) is a generative autoencoder first introduced by Kingma and Welling (2013) and differs by its lower-dimensional space z being non-deterministic (Figure 2). Given ϕ and θ the encoder and decoder parameters respectively, the projection of x into z is a Gaussian probability density noted $q_\phi(z|x)$. This prior knowledge is embedded into the objective function through the following regularization term:

$$D_{KL}(q_\phi(z|x)||p_\theta(z)) \tag{1}$$

which is Kullback-Leibler divergence between the encoder distribution and the expected distribution $p_\theta(z) \sim \mathcal{N}(0, I)$.[2]

[2]Inspired by an unpublished article by Quentin Fournier and Daniel Aloise.

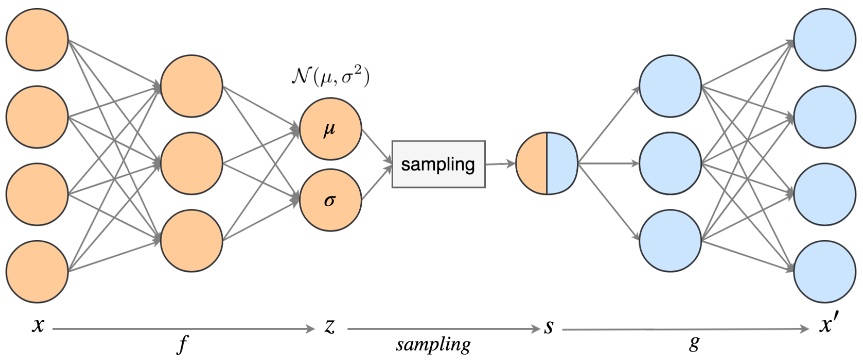


Figure 2: Framework of a variational autoencoder.

Finally, the objective function of a VAE is the sum of the reconstruction loss and the KL-divergence :

$$\mathcal{L}(\phi, \theta, x) = D_{KL}(q_\phi(z|x)||p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}(log\ p_\theta(x|z)) \tag{2}$$

2.2 Naive Bayes

Bayes classifier is a statistical classifier which predicts the probability of a given sample belongs to a particular class. While this is the best classifier, it is also difficult to use in practice.

Naive Bayes classifier is similar to the Bayes classifier but assumes that the effect of an attribute value on a given class is independent with regard to the values of the other attributes. This naive hypothesis is called “class conditional independence” and helps to simplify the calculation.

Naive Bayes is a simple yet very powerful classifier: although it is based on the *maximum a posteriori* (MAP) decision rule in a Bayesian environment, naive Bayes can easily be trained in a supervised fashion thanks to the *maximum likelihood.* In other words, one can work with the naive Bayes model without worrying about using Bayesian methods.

Let us explain in more detail how the naive Bayes classifier works:

1. Let T be a set of training observations with their class label. Suppose there are k distinct classes, $C_1, C_2, ..., C_k$. Each observation is represented by a n-dimensional vector, $X = x_1, x_2, ..., x_n$, describing the measured values of n attributes $A_1, A_2, ..., A_n$.

2. Given an observation X, the classifier predicts that X belongs to the class with the highest posterior probability conditional on X. More formally, X belongs to the class C_i if and only if:

$$P(C_i|X) > P(C_j|X) \ \ \text{for} j = 1, 2, ..., k, \ \ j \neq i. \tag{3}$$

   The class C_i for which $P(C_i|X)$ is maximal is called the posterior maximal hypothesis.

3. Note the Bayes theorem:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{4}$$

Since $P(X)$ is the same for all classes, only $P(X|C_i)P(C_i)$ has to be maximized. If the classes prior probabilities are unknown, one can generally assume that classes are equally probable $P(C_1) = P(C_2) = ... = P(C_k)$ and maximize only $P(X|C_i)$. Note that each class's prior probability can also be estimated as the frequency of the class:

$$P(C_i) = \frac{freq(C_i, T)}{|T|} \tag{5}$$

4. Given observations with many attributes, it would be computationally expensive to evaluate $P(X|C_i)$. The naive assumption of class conditional independence helps to speed up the evaluation of $P(X|C_i)P(C_i)$. Mathematically, the assumption can be written as follows:

   $$P(X|C_i) \approx \prod_{k=1}^{n} P(X_k|C_i) \tag{6}$$

   The probabilities $P(x_1|C_i), P(x_2|C_i), ..., P(x_n|C_i)$ can easily be estimated from the training set. Note that x_k refers to the value of attribute A_k for the sample X.

   (a) If A_k is categorical, then $P(x_k|C_i)$ is the number of samples of class C_i in T having the value x_k for attribute A_k divided by $freq(C_i, T)$, the number of observations of class C_i in T.

   (b) If A_k is continuous-valued, then we typically assume that the values have a Gaussian distribution with a mean μ and variance σ defined by:

   $$\frac{1}{\sqrt{2\pi\sigma}} exp\left(\frac{-(x-\mu)^2}{2\sigma}\right) \tag{7}$$

   μ_{C_i} and σ_{C_i} are evaluated as the mean and standard deviation of A_k's values for training samples of class C_i.

5. In order to predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts the class that maximizes $P(X|C_i)P(C_i)$.

2.3 Discriminant Analysis

2.3.1 Linear

The *linear discriminant analysis* (LDA) is one of the predictive discriminant techniques. LDA explains and predicts the class of an observation based on its characteristics measured using predictive variables. Similarly to naive Bayes, LDA is a parametric probabilistic method and classify a new observation x into one of the known K groups based on the probability $P(x \in k|x)$. That is, one needs to know the posterior probability of belonging to each class given the observation and then classify new observations as belonging to the class with the highest posterior probability.

Note that the Bayes theorem states that the posterior probability of an observation x belonging to a group k is:

$$P(x \in k|x) = \frac{\pi_k f(x_i|x \in k)}{\sum\limits_{l=1}^{k} \pi_l f(x_i|x \in l)} \tag{8}$$

Linear discriminant analysis assumes that observations from a group k follow a multivariate normal distribution with mean μ_k and covariance Σ. Note that the covariance matrix is shared across all groups. Assuming there is n attributes, the posterior probability is:

$$\begin{aligned} f(x|x \in k) &= f(x|\mu_k, \Sigma) && (9) \\ &= \left(\frac{1}{2\pi}\right)^{\frac{n}{2}} \times |\Sigma|^{-\frac{1}{2}} \times exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)\right) && (10) \end{aligned}$$

The prior probability $\pi_k = P(x \in k)$, which is the proportion of observations belonging to class k, is often estimated by $\pi_k = \frac{1}{K}$.

One is able to compute the posterior probability satisfying:

$$P(x \in k|x) \propto \pi_k f(x|\mu_k, \Sigma) \tag{11}$$

Hence:

$$P(x \in k|x) > P(x \in l|x) \Leftrightarrow \pi_k f(x|\mu_k, \Sigma) > \pi_l f(x|\mu_l, \Sigma) \tag{12}$$

Let us take the logarithm and substitute the probability density function by a multivariate normal distribution. After simplification, $P(x \in k|x) > P(x \in l|x)$ if and only if :

$$log(\pi_k) + x^T \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k > log(\pi_l) + x^T \Sigma^{-1}\mu_l - \frac{1}{2}\mu_l^T \Sigma^{-1}\mu_l \tag{13}$$

A linear model with regard to x emerges, hence the name “linear discriminate analysis”.

2.3.2 Quadratic

Quadratic discriminate analysis (QDA) is another kind of discriminant method. QDA differs from LDA only by not assuming equal covariance between classes. By applying the same reasoning as for LDA:

$$P(x \in k|x) > P(x \in l|x) \Leftrightarrow \pi_k f(x|\mu_k, \Sigma_k) > \pi_l f(x|\mu_l, \Sigma_l) \tag{14}$$

Implies that:

$$log(\pi_k) - \frac{1}{2}log(|\Sigma_k|) - \frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k) > log(\pi_l) - \frac{1}{2}log(|\Sigma_l|) - \frac{1}{2}(x-\mu_l)^T \Sigma_l^{-1}(x-\mu_l) \tag{15}$$

Note that no simplification arises. A quadratic form of x emerges, hence the name “quadratic discriminate analysis”.

3 Experiments

The purpose of this project is to measure the effect of violating – or respecting – the assumptions made by the three methods described in section 2. Luckily, all hypothesis can be met by making a stronger assumption: for a class k, observations must follow a unit Gaussian distribution $\mathcal{N}(0, I)$.

A variational autoencoder yields a projection whose each feature $z_i \sim \mathcal{N}(0, 1)$. Because each dimension (*i.e.* feature) is independent, this is equivalent to $z \sim \mathcal{N}(0, I)$. And because it is an unsupervised method, each class will also follow the same distribution. Moreover, one can assume that the latent representation z preserves as much information as possible since the decoder is able to reconstruct the input from z.

The rest of this section is organized as follows: section 3.1 presents the methodology. Section 3.2 describes the dataset and the preprocessing steps. Section 3.3 details the architecture of the variational autoencoder. Finally, section 3.4 presents the main results.

3.1 Methodology

Our project can be break down to four main steps:

1. Preprocess and analyze the data. In particular, we would like to verify that the features are not normally distributed in the original space.
2. Train a variational autoencoder. Analyze qualitatively and quantitatively the distribution of the validation set projection.
3. Evaluate naive Bayes, linear and quadratic discriminant analysis on the original space (*i.e.* when the assumptions are violated).
4. Evaluate the three methods on the learned representation. Because deep neural networks have many hyperparameters, one need to do a grid search to find a good solution.

This project has been realized in Python[3] with Jupyter Notebook[4].

3.2 Dataset

The data set used was released under the Open Data Commons licence by Pozzolo et al. (2015) and can be found at this address: `https://www.kaggle.com/mlg-ulb/creditcardfraud`.

The data set contains 284,807 anonymized credit card transactions labelled as fraudulent (1) or genuine (0). Note that there are only 492 (0.17%) fraudulent transactions. Each observation is composed of 30 attributes: Time, V1, V2, ..., V28, Amount.

As the scale of each feature is different, the observations have been standardized. Given x_{ij} the jth feature value of the ith observation, the standardize $\tilde{x_{ij}}$ is given by:

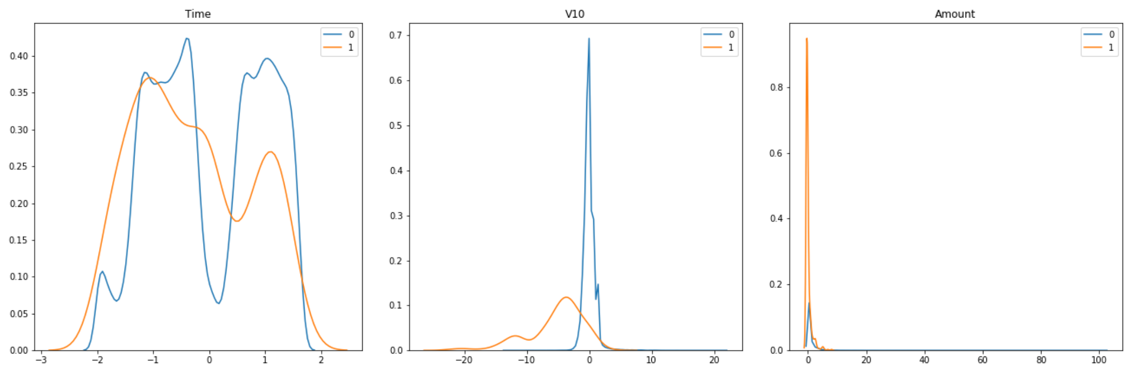
$$\tilde{x_{ij}} = \frac{x_{ij} - \bar{x_j}}{s_j} \tag{16}$$

With $\bar{x_j}$ the mean and s_j the standard deviation of the jth feature across the data set.

[3] `https://www.python.org`

[4] `https://jupyter.org`

Scipy's function *normaltest()* tests the null hypothesis that a sample comes from a normal distribution. When applied to the entire data set, every p-value is lower than 10^{-120}, so we can firmly conclude that no feature is normally distributed. Figure 3 shows the distribution of three features after standardization.



(a) Time. (b) V10. (c) Amount.

Figure 3: Distribution of three different features for each class as estimated by a Gaussian kernel density estimator.

Note that distributions are computed with a Gaussian kernel density estimator. The kernel size is determined by Scott's method, which is the default choice in scipy[5].

FInally, the data set was fairly split into a training set, a validation set and a test set of 180,000, 50,000 and 50,000 observations respectively. "Fairly split" means that the proportion of each class in each subset is the same as in the complete data set. Our VAE implementation requires to have equally sized mini-batch, hence the round number of observations in each subset.

3.3 Variational Autoencoder

Only the encoder will be presented as the decoder is symmetric to the encoder.

3.3.1 Encoder

The encoder is constructed by stacking layers as follows:

1. An input layer of dimension d_{in}.
2. An hidden layer of dimension $2d_{in}$. Projecting the data into a higher dimensional space can be useful to break non-linearity (similar to the kernel trick used by support vector machines).
3. A batch normalization layer which helps reduce the internal covariance shift[6] (Ioffe and Szegedy, 2015) and smooths the loss landscape (Santurkar et al., 2018). In practice, this layer allows for a faster training.
4. A non-linear activation function.

[5]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html`

[6]The distribution of each layer's input changes during training, as the parameters of the previous layers change. This phenomenon makes learning delicate.

5. An hidden layer of dimension $\frac{2d_{in}+d_h}{2}$ with d_h the latent representation dimension.

6. A batch normalization layer.

7. A non-linear activation function.

8. Two hidden layers: one for the vector of means μ and one for the vector of σ.

9. a sample layer that draws a projection from $\mathcal{N}(\mu, \sigma)$.

An accurate graphical representation of the VAE is presented in appendix (Figure 6).

3.3.2 Training

Let us look at the implementation of the objective function:

- The features are continuous-valued so the reconstruction error is measured by the mean square error between the input x and the output $\hat{x}$.
- The KL-divergence can be computed in closed form because both $p(z)$ and $q(z|x)$ are Gaussian:

$$\begin{aligned} D_{KL}(q_\phi(z|x)||p_\theta(z)) &= D_{KL}\left(\mathcal{N}(\mu(x), \Sigma(x))||\mathcal{N}(0,1)\right) & (17) \\ &= \frac{1}{2}\sum k\left(\Sigma(x) + \mu^2(x) - 1 - log\Sigma(x)\right) & (18) \end{aligned}$$

Finally, the objective function is a weighted sum of the two terms:

$$\mathcal{L}(x, z, \hat{x}) = \alpha MSE(x, \hat{x}) + (1-\alpha)\frac{1}{2}\sum k\left(\Sigma(x) + \mu^2(x) - 1 - log\Sigma(x)\right) \tag{19}$$

Deep neural network are often trained with vanilla stochastic gradient descent. As explained by Ruder (2016), one can speed up the learning phase by using an adaptive mini-batch gradient decent such as *Adam* (Kingma and Ba, 2014). The training was stopped when the objective function did not decrease by at least 0.01 in 5 iterations as measured on the validation set.

Most neural networks depend on many hyperparameters. Table 1 presents only the ones that were the subject of a grid search. Notably, we omitted the number of layers and their size as well as Adam's hyperparameters (learning rate, β_1, β_2 and decay).

Hyperparameter	Range	Grid search values
α	$[0, 1]$	$[0.25, 0.5, 0.75]$
mini-batch size	$[1, 50000]$	$[100, 1000]$
latent dimension d_h	$[1, \inf[$	$[10, 30, 50]$
activation function	C^1 functions	[relu, tanh]

Table 1: List of the main hyperparameters, their range and the values used for the grid search. *relu* stands for rectified linear unit and *tanh* for hyperbolic tangent.

3.4 Results

3.4.1 Projection Analysis

First, let us look qualitatively at the distribution of the learned projection. Figure 4 show the distribution of observations from the validation set.

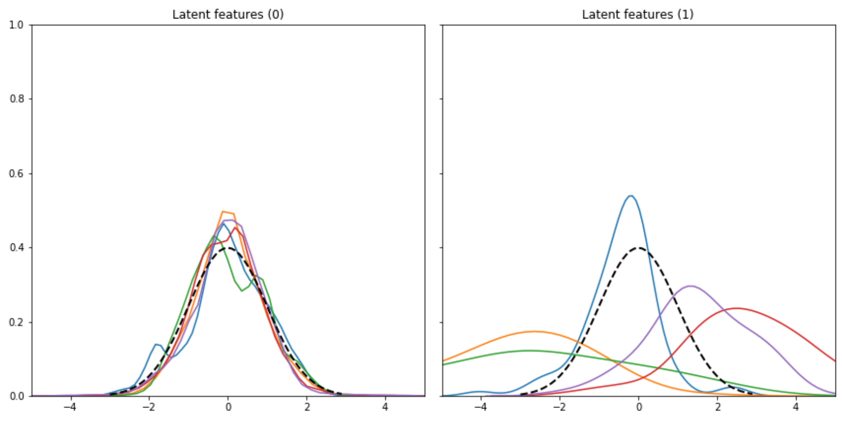


Figure 4: Distribution of the latent representation learned by the VAE. **Left**. Distribution of the class 0. **Right**. Distribution of the class 1. We hypothesize that the difference in quality comes from the small amount of observation from class 1. One could add a class weight during training to minimize the difference.

The Q-Q plot gives us more insight about the normality of our representation (Figure 5).

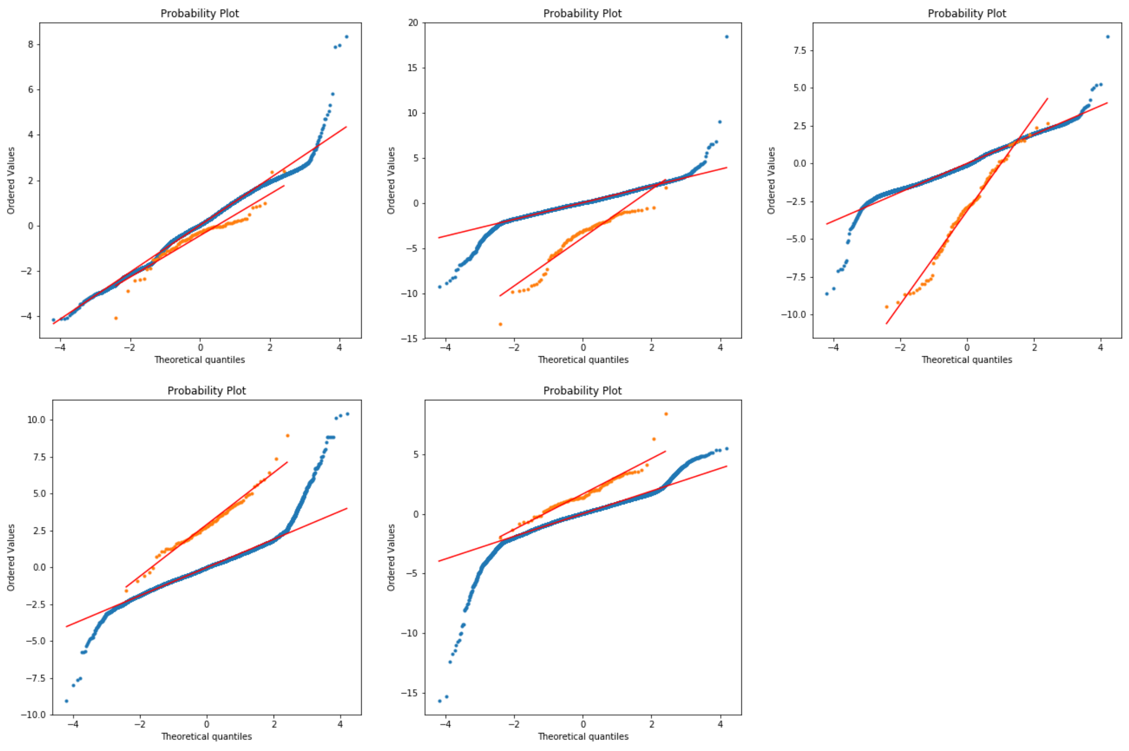


Figure 5: Q-Q plot of the learned representation. Blue and orange curves represent the classes 0 and 1 respectively.

One can clearly see that both extremities of the blue curve separate in a spectacular way from the reference line. It is hard to accept the normality of the class 0 under this observation.

Except for the middle-top plot, the orange curve seems to follow the reference line. One would conclude that the hypothesis of normality can be accepted for class 1.

For the rest of the project, we are going to assume that the normality hypothesis is verified asymptotically, which is ensured by the central limit theorem. Especially, the central limit theorem works with large samples even if there is difference in proportion between the two classes.

3.4.2 Linear Discriminant Analysis

Linear discriminant analysis performs well directly on the original space (Table 2). LDA yields slightly better results on the learned projection. We hypothesize that the impact of the projection is negligible because it does not verify well enough the assumptions.

Space	Accuracy (%)	Precision (%)	Recall (%)
Original	99.934	77.011	83.750
VAE(50, 0.50, tanh, 100)	**99.938**	77.907	84.810
VAE(30,0.50, tanh, 1000)	99.936	**79.070**	82.927
VAE(30, 0.75, tanh, 100)	99.936	75.581	**85.526**

Table 2: Accuracy, precision and recall of LDA on the original space and the best projection according to the 3 metrics. Note that the difference is not significant. For the complete list of results, see appendix B.1.

3.4.3 Quadratic Discriminant Analysis

Quadratic discriminant analysis preforms a lot better on the learned representation than on the original space (Table 3). In fraud detection context, one would typically want a high recall rate rather than a high accuracy, which can be greatly by using a VAE.

Space	Accuracy (%)	Precision (%)	Recall (%)
Original	97.426	**86.207**	5.556
VAE(10, 0.25, tanh, 100)	**99.778**	77.907	**42.138**
VAE(30, 0.25, relu, 1000)	97.972	83.721	6.716

Table 3: Accuracy, precision and recall of QDA on the original space and the best projection according to the 3 metrics. Note that the same model yields both the best accuracy and the best recall. Also, the learned projection did not allows for a better precision with QDA. For the complete list of results, see appendix B.2.

In the complete list of results presented in appendix B.2, we can see that certain set of hyperparameters performs very poorly. We hypothesize the main cause to be that only one network is trained per set of hyperparameters. Networks parameters are randomly initialized, and the mini-batches are randomly created, making the training highly non deterministic. Even with batch normalization and Adam optimizer, one could expect to have a few “bad runs”. One way to reduce this phenomenon would be to use cross validation. However it is very expensive.

3.4.4 Naive Bayes

The analysis of the naive Bayes classifier is the same as for QDA. The learned representation allows to increase greatly the recall rate.

Space	Accuracy (%)	Precision (%)	Recall (%)
Original	97.818	82.759	6.272
VAE(10, 0.25, tanh, 100)	**99.786**	77.907	**43.226**
VAE(50, 0.50, tanh, 100)	97.476	**83.721**	5.455

Table 4: Accuracy, precision and recall of naive Bayes on the original space and the best projection according to the 3 metrics. Note that the same model yields both the best accuracy and the best recall. For the complete list of results, see appendix B.2.

4 Conclusion

To summarize, we propose the use of a variational autoencoder to justify the systematic use of naive Bayes, linear and quadratic discriminant analysis. When apply on a real data set, we saw that the learned projection helped greatly improve the recall rate of both QDA and naive Bayes. LDA, while already accurate on the original space, did not yield significantly better results on the learned projection. We hypothesize the cause to be the quality of the learned representation: one of the assumption is violated as the variance vary between classes.

Class 1 (fraudulent) is under represented which make it harder for the network to learn a good projection. This could be addressed by adding a class weight inversely proportional to the number of observation in each class. Also, one could expect to have better results by training multiple networks for each set of hyperparameters.

References

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science (80-.).*, 313(5786):504–507.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.0.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

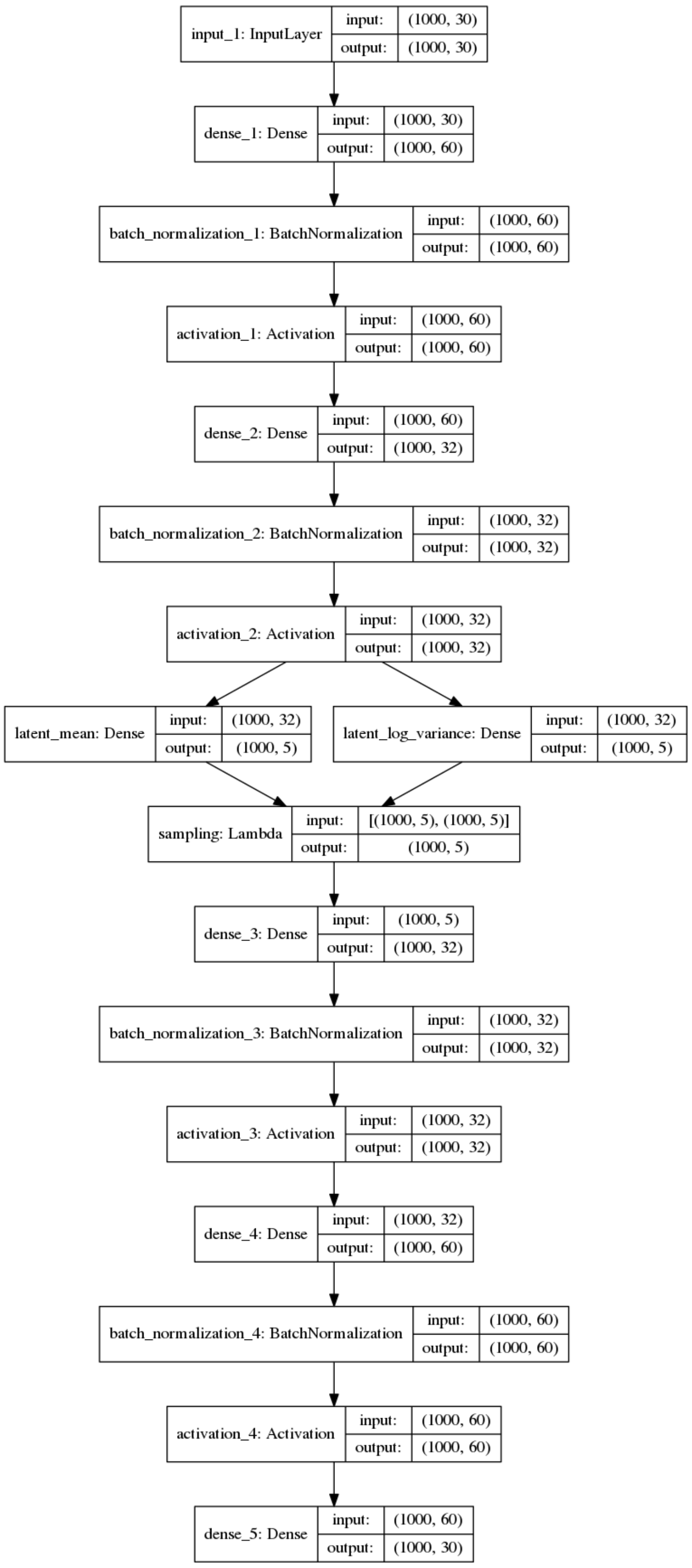
Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *CoRR*, abs/1312.6.

Pozzolo, A. D., Caelen, O., Johnson, R. A., and Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *IEEE Symposium Series on Computational Intelligence, SSCI 2015, Cape Town, South Africa, December 7-10, 2015*, pages 159–166.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv e-prints.*

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How Does Batch Normalization Help Optimization? In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Adv. Neural Inf. Process. Syst. 31*, pages 2488–2498. Curran Associates, Inc.

A Variational Autoencoder Architecture



input_1: InputLayer | input: (1000, 30) | output: (1000, 30)
dense_1: Dense | input: (1000, 30) | output: (1000, 60)
batch_normalization_1: BatchNormalization | input: (1000, 60) | output: (1000, 60)
activation_1: Activation | input: (1000, 60) | output: (1000, 60)
dense_2: Dense | input: (1000, 60) | output: (1000, 32)
batch_normalization_2: BatchNormalization | input: (1000, 32) | output: (1000, 32)
activation_2: Activation | input: (1000, 32) | output: (1000, 32)
latent_mean: Dense | input: (1000, 32) | output: (1000, 5)
latent_log_variance: Dense | input: (1000, 32) | output: (1000, 5)
sampling: Lambda | input: [(1000, 5), (1000, 5)] | output: (1000, 5)
dense_3: Dense | input: (1000, 5) | output: (1000, 32)
batch_normalization_3: BatchNormalization | input: (1000, 32) | output: (1000, 32)
activation_3: Activation | input: (1000, 32) | output: (1000, 32)
dense_4: Dense | input: (1000, 32) | output: (1000, 60)
batch_normalization_4: BatchNormalization | input: (1000, 60) | output: (1000, 60)
activation_4: Activation | input: (1000, 60) | output: (1000, 60)
dense_5: Dense | input: (1000, 60) | output: (1000, 30)

Figure 6: Graphical representation of the VAE.

B Complete Grid Search Results

B.1 Linear Discriminant Analysis

```
(latent dim, alpha, activation, batch size): accuracy  | precision (1)| recall (1)
(    10    , 0.25 ,    tanh    ,    100    ):  99.934% |   77.907%    |  82.716%
(    10    , 0.25 ,    tanh    ,    1000   ):  99.896% |   75.581%    |  67.708%
(    10    , 0.25 ,    relu    ,    100    ):  99.930% |   74.419%    |  83.117%
(    10    , 0.25 ,    relu    ,    1000   ):  99.928% |   76.744%    |  80.488%
(    10    , 0.50 ,    tanh    ,    100    ):  99.914% |   79.070%    |  73.118%
(    10    , 0.50 ,    tanh    ,    1000   ):  99.870% |   79.070%    |  59.130%
(    10    , 0.50 ,    relu    ,    100    ):  99.926% |   76.744%    |  79.518%
(    10    , 0.50 ,    relu    ,    1000   ):  99.908% |   75.581%    |  72.222%
(    10    , 0.75 ,    tanh    ,    100    ):  99.880% |   76.744%    |  62.264%
(    10    , 0.75 ,    tanh    ,    1000   ):  99.904% |   76.744%    |  70.213%
(    10    , 0.75 ,    relu    ,    100    ):  99.898% |   55.814%    |  78.689%
(    10    , 0.75 ,    relu    ,    1000   ):  99.880% |   76.744%    |  62.264%
(    30    , 0.25 ,    tanh    ,    100    ):  99.918% |   76.744%    |  75.862%
(    30    , 0.25 ,    tanh    ,    1000   ):  99.824% |   75.581%    |  49.242%
(    30    , 0.25 ,    relu    ,    100    ):  99.934% |   76.744%    |  83.544%
(    30    , 0.25 ,    relu    ,    1000   ):  99.884% |   72.093%    |  64.583%
(    30    , 0.50 ,    tanh    ,    100    ):  99.930% |   74.419%    |  83.117%
(    30    , 0.50 ,    tanh    ,    1000   ):  99.936% |   79.070%    |  82.927%
(    30    , 0.50 ,    relu    ,    100    ):  99.924% |   75.581%    |  79.268%
(    30    , 0.50 ,    relu    ,    1000   ):  99.934% |   74.419%    |  85.333%
(    30    , 0.75 ,    tanh    ,    100    ):  99.936% |   75.581%    |  85.526%
(    30    , 0.75 ,    tanh    ,    1000   ):  99.916% |   67.442%    |  80.556%
(    30    , 0.75 ,    relu    ,    100    ):  99.928% |   75.581%    |  81.250%
(    30    , 0.75 ,    relu    ,    1000   ):  99.866% |   75.581%    |  58.559%
(    50    , 0.25 ,    tanh    ,    100    ):  99.938% |   79.070%    |  83.951%
(    50    , 0.25 ,    tanh    ,    1000   ):  99.934% |   79.070%    |  81.928%
(    50    , 0.25 ,    relu    ,    100    ):  99.932% |   76.744%    |  82.500%
(    50    , 0.25 ,    relu    ,    1000   ):  99.936% |   76.744%    |  84.615%
(    50    , 0.50 ,    tanh    ,    100    ):  99.938% |   77.907%    |  84.810%
(    50    , 0.50 ,    tanh    ,    1000   ):  99.936% |   79.070%    |  82.927%
(    50    , 0.50 ,    relu    ,    100    ):  99.936% |   79.070%    |  82.927%
(    50    , 0.50 ,    relu    ,    1000   ):  99.916% |   68.605%    |  79.730%
(    50    , 0.75 ,    tanh    ,    100    ):  99.936% |   79.070%    |  82.927%
(    50    , 0.75 ,    tanh    ,    1000   ):  99.936% |   77.907%    |  83.750%
(    50    , 0.75 ,    relu    ,    100    ):  99.934% |   76.744%    |  83.544%
(    50    , 0.75 ,    relu    ,    1000   ):  99.912% |   63.953%    |  80.882%
```

B.2 Quadratic Discriminant Analysis

```
(latent dim, alpha, activation, batch size): accuracy  | precision (1)| recall (1)
(    10    , 0.25 ,    tanh   ,    100    ):  99.778%  |   77.907%    |  42.138%
(    10    , 0.25 ,    tanh   ,    1000   ):  99.616%  |   77.907%    |  27.917%
(    10    , 0.25 ,    relu   ,    100    ):  99.678%  |   76.744%    |  31.884%
(    10    , 0.25 ,    relu   ,    1000   ):  99.724%  |   75.581%    |  35.714%
(    10    , 0.50 ,    tanh   ,    100    ):  99.730%  |   79.070%    |  36.757%
(    10    , 0.50 ,    tanh   ,    1000   ):  99.502%  |   79.070%    |  22.742%
(    10    , 0.50 ,    relu   ,    100    ):  99.706%  |   79.070%    |  34.518%
(    10    , 0.50 ,    relu   ,    1000   ):  99.676%  |   79.070%    |  32.075%
(    10    , 0.75 ,    tanh   ,    100    ):  99.428%  |   79.070%    |  20.238%
(    10    , 0.75 ,    tanh   ,    1000   ):  98.806%  |   79.070%    |  10.510%
(    10    , 0.75 ,    relu   ,    100    ):  99.414%  |   68.605%    |  18.154%
(    10    , 0.75 ,    relu   ,    1000   ):  99.460%  |   77.907%    |  21.069%
(    30    , 0.25 ,    tanh   ,    100    ):  97.334%  |   83.721%    |   5.176%
(    30    , 0.25 ,    tanh   ,    1000   ):  97.038%  |   83.721%    |   4.678%
(    30    , 0.25 ,    relu   ,    100    ):  97.414%  |   83.721%    |   5.329%
(    30    , 0.25 ,    relu   ,    1000   ):  97.972%  |   83.721%    |   6.716%
(    30    , 0.50 ,    tanh   ,    100    ):  97.682%  |   82.558%    |   5.844%
(    30    , 0.50 ,    tanh   ,    1000   ):  97.538%  |   83.721%    |   5.586%
(    30    , 0.50 ,    relu   ,    100    ):  97.490%  |   83.721%    |   5.484%
(    30    , 0.50 ,    relu   ,    1000   ):  97.698%  |   82.558%    |   5.882%
(    30    , 0.75 ,    tanh   ,    100    ):  97.560%  |   83.721%    |   5.634%
(    30    , 0.75 ,    tanh   ,    1000   ):  97.560%  |   83.721%    |   5.634%
(    30    , 0.75 ,    relu   ,    100    ):  97.556%  |   83.721%    |   5.625%
(    30    , 0.75 ,    relu   ,    1000   ):  98.748%  |   82.558%    |  10.411%
(    50    , 0.25 ,    tanh   ,    100    ):  98.344%  |   83.721%    |   8.126%
(    50    , 0.25 ,    tanh   ,    1000   ):  98.340%  |   83.721%    |   8.108%
(    50    , 0.25 ,    relu   ,    100    ):  98.620%  |   81.395%    |   9.409%
(    50    , 0.25 ,    relu   ,    1000   ):  99.274%  |   80.233%    |  16.627%
(    50    , 0.50 ,    tanh   ,    100    ):  98.332%  |   82.558%    |   7.978%
(    50    , 0.50 ,    tanh   ,    1000   ):  98.582%  |   81.395%    |   9.174%
(    50    , 0.50 ,    relu   ,    100    ):  98.048%  |   81.395%    |   6.796%
(    50    , 0.50 ,    relu   ,    1000   ):  99.332%  |   80.233%    |  17.876%
(    50    , 0.75 ,    tanh   ,    100    ):  98.662%  |   82.558%    |   9.793%
(    50    , 0.75 ,    tanh   ,    1000   ):  98.336%  |   82.558%    |   7.995%
(    50    , 0.75 ,    relu   ,    100    ):  98.676%  |   82.558%    |   9.889%
(    50    , 0.75 ,    relu   ,    1000   ):  99.424%  |   81.395%    |  20.468%
```

B.3 Naive Bayes

```
(latent dim, alpha, activation, batch size): accuracy  | precision (1)| recall (1)
(    10    , 0.25 ,    tanh    ,    100    ):  99.786% |   77.907%    |  43.226%
(    10    , 0.25 ,    tanh    ,    1000   ):  99.564% |   79.070%    |  25.373%
(    10    , 0.25 ,    relu    ,    100    ):  99.334% |   76.744%    |  17.414%
(    10    , 0.25 ,    relu    ,    1000   ):  99.358% |   66.279%    |  16.332%
(    10    , 0.50 ,    tanh    ,    100    ):  99.740% |   79.070%    |  37.778%
(    10    , 0.50 ,    tanh    ,    1000   ):  99.672% |   76.744%    |  31.429%
(    10    , 0.50 ,    relu    ,    100    ):  99.546% |   75.581%    |  23.985%
(    10    , 0.50 ,    relu    ,    1000   ):  99.330% |   79.070%    |  17.662%
(    10    , 0.75 ,    tanh    ,    100    ):  99.716% |   77.907%    |  35.263%
(    10    , 0.75 ,    tanh    ,    1000   ):  99.352% |   80.233%    |  18.351%
(    10    , 0.75 ,    relu    ,    100    ):  99.544% |   60.465%    |  21.138%
(    10    , 0.75 ,    relu    ,    1000   ):  99.348% |   76.744%    |  17.742%
(    30    , 0.25 ,    tanh    ,    100    ):  97.652% |   82.558%    |   5.772%
(    30    , 0.25 ,    tanh    ,    1000   ):  97.904% |   81.395%    |   6.352%
(    30    , 0.25 ,    relu    ,    100    ):  97.090% |   83.721%    |   4.759%
(    30    , 0.25 ,    relu    ,    1000   ):  97.490% |   81.395%    |   5.348%
(    30    , 0.50 ,    tanh    ,    100    ):  97.622% |   81.395%    |   5.632%
(    30    , 0.50 ,    tanh    ,    1000   ):  97.482% |   82.558%    |   5.399%
(    30    , 0.50 ,    relu    ,    100    ):  97.366% |   77.907%    |   4.908%
(    30    , 0.50 ,    relu    ,    1000   ):  97.944% |   81.395%    |   6.470%
(    30    , 0.75 ,    tanh    ,    100    ):  98.082% |   77.907%    |   6.653%
(    30    , 0.75 ,    tanh    ,    1000   ):  98.032% |   76.744%    |   6.408%
(    30    , 0.75 ,    relu    ,    100    ):  98.042% |   79.070%    |   6.608%
(    30    , 0.75 ,    relu    ,    1000   ):  97.596% |   76.744%    |   5.288%
(    50    , 0.25 ,    tanh    ,    100    ):  97.438% |   82.558%    |   5.310%
(    50    , 0.25 ,    tanh    ,    1000   ):  97.476% |   81.395%    |   5.319%
(    50    , 0.25 ,    relu    ,    100    ):  97.904% |   81.395%    |   6.352%
(    50    , 0.25 ,    relu    ,    1000   ):  97.876% |   80.233%    |   6.194%
(    50    , 0.50 ,    tanh    ,    100    ):  97.476% |   83.721%    |   5.455%
(    50    , 0.50 ,    tanh    ,    1000   ):  97.548% |   81.395%    |   5.469%
(    50    , 0.50 ,    relu    ,    100    ):  97.208% |   83.721%    |   4.952%
(    50    , 0.50 ,    relu    ,    1000   ):  97.576% |   80.233%    |   5.459%
(    50    , 0.75 ,    tanh    ,    100    ):  98.016% |   80.233%    |   6.609%
(    50    , 0.75 ,    tanh    ,    1000   ):  97.960% |   81.395%    |   6.518%
(    50    , 0.75 ,    relu    ,    100    ):  97.138% |   81.395%    |   4.714%
(    50    , 0.75 ,    relu    ,    1000   ):  97.746% |   81.395%    |   5.927%
```