# Two Applications of ML in Finance

## Coding Exercises

*Sebastian Becker* *and* *Patrick Cheridito*

RiskLab, ETH Zurich

**ETH Zurich, April 9, 2021**

## I) Estimating Default Risk

**What we are going to do**

1. Simulate two artificial data sets with a *logistic model* and a *non-linear generalization*
2. Apply *logistic regression* to both data sets
3. Improve the performance on the *non-linear data set* with the help of *neural networks*
4. Run a simple *P&L analysis* on different loan portfolios

**The toolset**

- Google Colab `https://colab.research.google.com`
- Python with Numpy, Sklearn, Keras, Tensoflow

## Simulation of the two datasets

- Simulation of the *features*
  - $x_1$ = age *uniformly distributed over* $[18, 80]$
  - $x_2$ = monthly income in CHF 1000 *uniformly distributed over* $[4, 30]$
  - $x_3$ = salaried/self-employed *Bernoulli distributed over* $\{0, 1\}$ *with probabilities 0.9 and 0.1*

  (We generate 80,000 *training samples* and 20,000 test samples of $x = (x_1, x_2, x_3)$)

- In the *logistic model* we set the *"true" default probability* equal to

$$p_1(x) = \psi\left(1 + 0.33x_1 - 3.5x_2 + 3x_3\right)$$

  In the *non-linear logistic model* we set it equal to

$$p_2(x) = \psi\left(-1.25 + 2.5\left[\mathbb{1}_{(-\infty,25)}(x_1) + \mathbb{1}_{(75,\infty)}(x_1)\right] - 0.25x_2 + x_3\right)$$

- For every $x = (x_1, x_2, x_3)$ we run an independent *Bernoulli experiment* and attach to $x$ the *label*

$$y = \begin{cases} 1 & \text{with probability } p_i(x) \\ 0 & \text{with probability } 1 - p_i(x) \end{cases} , \qquad i = 1, 2$$

- The empirical default rate is about 5% on both *data sets* $i = 1, 2$

## Example dataset

| age (years) | income (K) | self-employed (y/n) | default (y/n) |
|---|---|---|---|
| 55.8 | 24.6 | 1 | 0 |
| 34.8 | 13.0 | 0 | 0 |
| 57.6 | 9.1 | 0 | 0 |
| 53.2 | 11.2 | 0 | 0 |
| 78.7 | 5.8 | 0 | **1** |
| 46.0 | 14.3 | 0 | 0 |
| 19.5 | 21.8 | 0 | 0 |
| 18.7 | 9.0 | 1 | 0 |
| 66.9 | 6.2 | 0 | **1** |
| 35.9 | 13.5 | 0 | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ |

## Fitting a logistic regression model to both datasets

We use a numerical method (lbfgs) to estimate the coefficients from the two *data sets*

Standard *logistic regression* for the ...

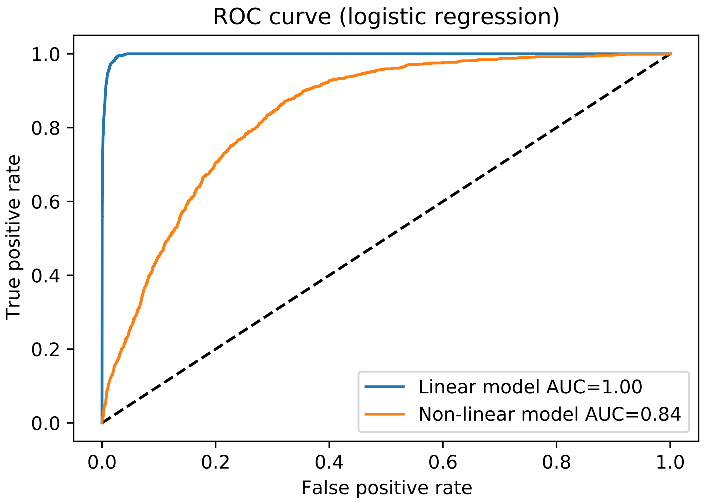
- *linear data set*

$$\hat{p}_1(x) = f_\theta(x) = \psi(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)$$

- and the *non-linear data set*

$$\hat{p}_2(x) = f_{\theta'}(x) = \psi(\theta_0^{'} + \theta_1^{'} x_1 + \theta_2^{'} x_2 + \theta_3^{'} x_3)$$

## Comparison of the logistic regression applied to the two datasets

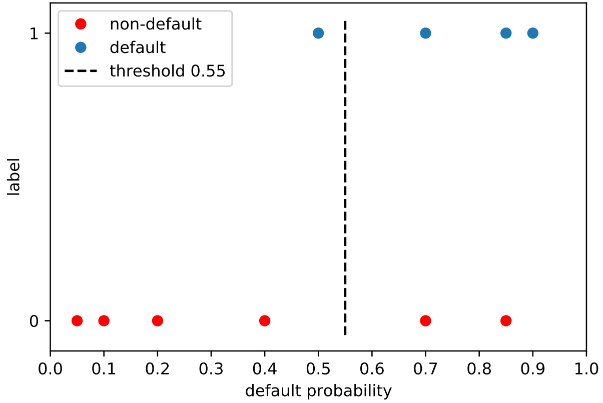


AUC = Area under the curve

## A closer look at ROC curves

For every *decision threshold* we calculate

- the true positive rate (TPR): $\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$,
- the false positive rate (FPR): $\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}}$,
- and plot TPR against FPR $(\mathrm{TPR} = 0.6, \mathrm{FPR} = 0.2)$



## Fitting a neural network to the non-linear dataset

- Standard *logistic regression* can only fit linear logits!
- One could use a more sophisticated method like random *forests, gradient boosting, SVMs, or neural networks ...*
- We fit a *neural network with two hidden layers* and train it with a variant of the *stochastic gradient descent* optimizer:

$$\hat{p}_2(x) = f_{\hat{\theta}}(x) = \psi \circ A_3 \circ \rho \circ A_2 \circ \rho \circ A_1,$$

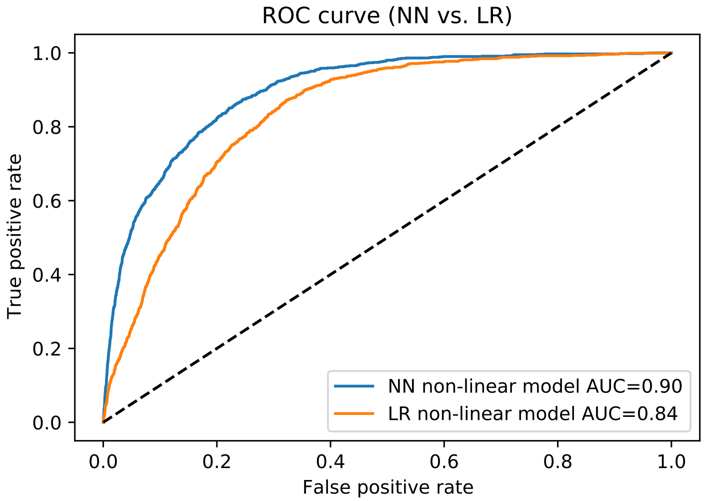
  where

$$A_1 \colon \mathbb{R}^3 \to \mathbb{R}^{32},\ A_2 \colon \mathbb{R}^{32} \to \mathbb{R}^{32},\ A_3 \colon \mathbb{R}^{32} \to \mathbb{R} \quad \text{are affine and} \quad \rho(x) = \max\{x, 0\} \quad \text{(ReLU)}$$

- Loss function: *total deviance (negative conditional log-likelihood)*

$$\theta \mapsto \sum_{j=1}^{J} -y^j \log f_\theta(x^j) - (1 - y^j) \log\left(1 - f_\theta(x^j)\right)$$

## Comparison of the NN model against logistic regression in the non-linear case



## P&L analysis in the non-linear case

- We consider a loan portfolio of 5000 loans with a loan amount of CHF 1000
- The 5000 borrowers are sampled from the *non-linear test data set*
- To estimate the distribution of the P&L, we simulate the following 50,000 times:

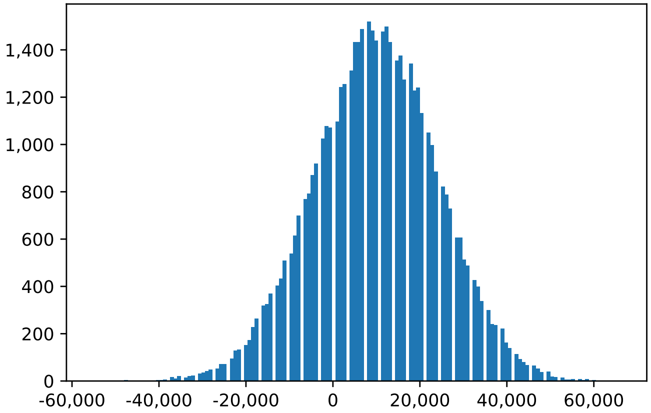
  **Scenario 1** We lend money with an interest rate of 5.5% to all 5000 candidates

  **Scenario 2** We lend money with an interest rate of 1.5% to everybody with a *predicted default probability* $\hat{p}_i(x^j) \leq 5\%$ according to the *logistic regression* $(i = 1)$ or the *neural network* $(i = 2)$

- In both scenarios, we
  - plot *histograms* of the *P&L*
  - estimate the *expected P&L*
  - compute the *99%-Value-at-Risk* = negative of the 1%-quantile of the P&L

## P&L analysis of Scenario 1

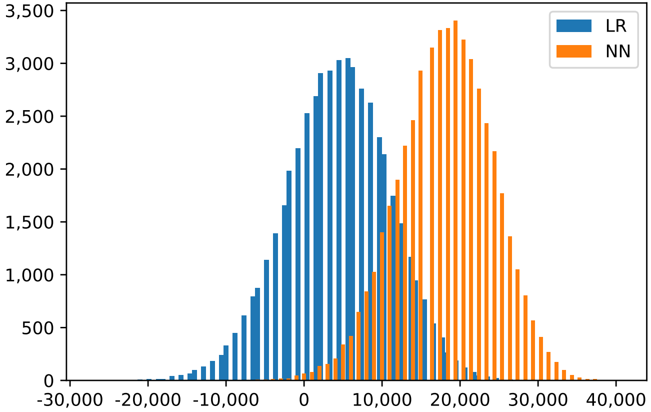
Expected P&L = 9863;     99%-VaR = 23,565



## P&L analysis of Scenario 2

*Logistic regression:* expected P&L = 4298;    99%-VaR = 11,855

*Neural network:* expected P&L = 18,154;    99%-VaR = −3000



## Remarks ...

- The artificial data sets are not very realistic ... real data sets can have several hundred features
- Logistic regression is still the industry standard. Why?

  Explainability, regulation

- Could the results of the logistic regression on the non-linear data set be improved?

  Yes, with clever feature engineering!

- Can other ML methods compete with neural networks on the non-linear data set?

  Yes, even a simple decision tree can solve this task quite well!

## Some ideas how the code can be modified ...

1. Choose numbers $J, K \geq 5000$ and simulate $J + K$ vectors $x^j = (x_1^j, x_2^j, x_3^j) \in \mathbb{R}^3$ with
   - $x_1^j$ = age in $[18, 80]$
   - $x_2^j$ = monthly income in CHF 1000 in $[4, 30]$
   - $x_3^j$ = salaried/self-employed in $\{0, 1\}$

   Let $\xi^j, j = 1, \ldots, J + K$ be independent random variables that are uniformly distributed on $(0, 1)$ and $\psi \colon \mathbb{R} \to (0, 1)$ the logistic (or sigmoid) function given by

   $$\psi(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$

   Consider a function $p \colon \mathbb{R}^3 \to (0, 1)$ of the form $\quad p(x) = \psi\left(a_0 + a_1 |x_1 - 50| + a_2 x_2 + a_3 x_3\right)$

   and generate an artificial data set $(x^j, y_1^j), j = 1, \ldots, J + K$ by setting

   $$y^j = \begin{cases} 1 & \text{if } \xi^j \leq p(x^j) \\ 0 & \text{otherwise.} \end{cases}$$

   a) Choose $a_0, a_1, a_2, a_3 \in \mathbb{R}$ such that approximately 5% of all $y^j, j = 1, \ldots, J$, are 1.

   b) “Learn" $\hat{p}_1 \colon \mathbb{R}^3 \to \mathbb{R}$ on the *training data* $(x^j, y^j), j = 1, \ldots, J$, with *logistic regression.*

   c) Which fraction of $\left\{y^j : \hat{p}_1(x^j) \in [3\%, 4\%], j = J + 1, \ldots, J + K\right\}$ is 1?

   d) “Learn" $\hat{p}_2 \colon \mathbb{R}^3 \to \mathbb{R}$ from the *training data* $(x^j, y^j), j = 1, \ldots, J$, with a *neural network.*

   e) Which fraction of $\left\{y^j : \hat{p}_2(x^j) \in [3\%, 4\%], j = J + 1, \ldots, J + K\right\}$ is 1?

2. Find *good risks* in the *test data set* based on the *features* $x^j$, $j = J+1, \ldots, J+K$, to form a portfolio of loans, all in the amount of CHF 1000 with interest rate $4\%$.

   a) Estimate the expected P&L.

   b) Estimate the variance of the P&L.

   c) Estimate the $99\%$-VaR of the P&L (= negative of the $1\%$-quantile)

## II) Pricing and Hedging American-Style Derivatives

**What does the sample code do?**

1. Simulate stock prices $S^i_{n/3} = s^i_0 \exp\left([r - \delta_i - \sigma_i^2/2]n/3 + \sigma_i W^i_{n/3}\right)$ , $i = 1, 2, \ldots, 5$ for

$$s^i_0 = 100; \quad r = 5\%; \quad \delta_i = 10\%; \quad \sigma_i = 20\% \quad n = 0, 1, 2, 3$$

2. Find an optimal stopping time for the problem

$$\sup_{\tau \in \{t_1, \ldots, T\}} \mathbb{E}\left[e^{-r\tau}\left(\max_{1 \le i \le d} S^i_\tau - K\right)^+\right] \quad \text{for } K = 100$$

*It is not optimal to stop at 0. So one only has to make a stopping decision at $t_1$ and $t_2$!*

3. Estimate a lower bound $\hat{L}$

4. Estimate an upper bound $\hat{U}$

## Simulation of stock prices

We create the sample paths on the fly and do not distinguish between *training set and test set!*

**Training and estimation of the lower bound:**

- Create mini-batches of geometric Brownian motion sample paths

$$S^i_{n/3} = s^i_0 \exp\Big([r - \delta_i - \sigma_i^2/2]n/3 + \sigma_i W^i_{n/3}\Big)\,,\ i = 1, 2, \ldots, 5, \ \text{ for}$$

$$s^i_0 = 100; \quad r = 5\%; \quad \delta_i = 10\%; \quad \sigma_i = 20\% \quad n = 0, 1, 2, 3$$

**Estimation of the upper bound:**

- For every sample of $S^i_{n/3}$, $i = 1, 2, \ldots, 5$, $n = 0, 1, 2$, we need a set of independent continuation prices

$$\hat{S}^i_{j/3} = S^i_{n/3} \exp\Big([r - \delta_i - \sigma_i^2/2](j - n)/3 + \sigma_i(\hat{W}^i_{j/3} - \hat{W}^i_{n/3})\Big), \quad j = n + 1, \ldots, 3$$

## The neural network architecture

- For the *training* we use neural networks $F^\theta \colon \mathbb{R}^d \to (0,1)$ of the form

$$F^\theta = \psi \circ a_3^\theta \circ \rho \circ a_2^\theta \circ \rho \circ a_1^\theta \quad \text{for} \quad \psi(x) = \frac{e^x}{1+e^x}.$$

- Once trained, we switch back to hard stopping decision $f^\theta \colon \mathbb{R}^d \to \{0,1\}$ of the form

$$f^\theta(x) = 1_{[0,\infty)} \circ a_3^\theta \circ \rho \circ a_2^\theta \circ \rho \circ a_1^\theta(x).$$

- *Batch normalization* is applied after the *affine transormations* before activation. We also use it to normalize the inputs and the outputs of the neural networks.

- The sample codes contains two different versions of neural networks
  1. one uses two separate neural networks to model the stopping decisions at times 1 and 2
  2. the other combines these two networks

  Mathematically the two approaches are equivalent, but 2 is optimized for parallel computation and runs faster on GPUs. This comes in handy for larger problems!

## Batch normalization (Ioffe & Szegedy 2015)

- Avoids unstable gradients
- Reduces the effect of bad initialization
- Allows for larger learning rates and hence faster training

**How does it work?**

- Calculate mean $\mu$ and variance $\sigma^2$ of a mini-batch and compute a moving average of mean and variance during the training
- Normalize $\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$, where $\varepsilon > 0$ is a small number to avoid division by 0
- Scale and shift: $y_i = \gamma \hat{x}_i + \beta$, where $\gamma$ and $\beta$ are trainable variables
- Use the moving averages after the training

## The optimization problem in the sample code

We could train separate neural network recursively backwards in time. But instead, we train all neural networks *simultaneously!*

The loss function is then of the following form:

$$\text{loss} = -\sum_{n=1}^{2} g(n, S_n)F^{\theta_n}(S_n, g(n, S_n)) + g(\tau_{n+1}, S_{\tau_{n+1}})(1 - F^{\theta_n}(S_n, g(n, S_n)))$$

where

$$\tau_3 = 3$$

and

$$\tau_{n+1} = \sum_{m=n+1}^{3} m f^{\theta_m}(S_m, g(m, S_m)) \prod_{j=n+1}^{m-1} (1 - f^{\theta_j}(S_j, g(j, S_j))) \quad \text{for } n = 1$$

## Training and estimation of the lower and upper bound

- The neural networks are trained with the Adam optimizer for 5000 steps with an initial learning rate of 0.01 and $\varepsilon = 0.1$ and a batch size of 8192. After every 1500 training steps the learning rate is divided by 10.

- The parameters above can be tuned to save training steps and speed up the training. A smaller batch size is also possible.

- The lower bound is estimated with 4,096,000 sample paths and yields a price of 18.689 with a one-sided $97.5\%$ CI of $[18.676, \infty)$.

- The upper bound is estimated with 2048 samples where for each sample path we simulate in each time step another 2048 nested sample paths. This yields an estimate of 18.703 with a one-sided $97.5\%$ CI of $(-\infty, 18.716]$. The number of samples was chosen such that the confidence margin came out about the same as the lower bound.

- The resulting price estimate $\hat{V}$ is 18.696 with a two-sided $95\%$ CI of $[18.676, 18.716]$.

## Hedging

- In addition to the prices simulated in the optimal stopping example, one also needs dicounted dividend adjusted prices on a finer grid $P^i_{u_m} = p^i_m\left(W^i_{u_m}\right) = s^i_0 \exp\left(\sigma_i W^i_{u_m} - \sigma_i^2 u_m/2\right)$ for $m = 0, 1, \dots, NM$

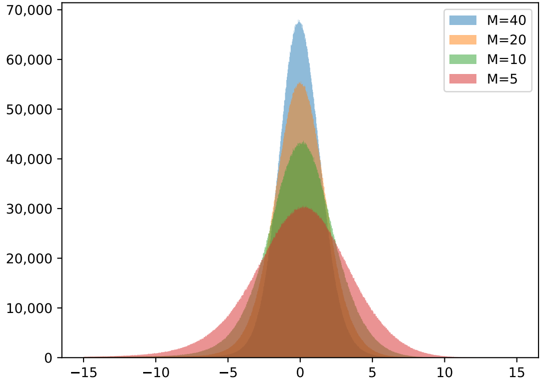
- For every hedging rebalancing time one needs a neural network. The problem can become very large!

- We also need the trained neural networks from the optimal stopping problem to be able to evaluate the optimal payoff $g(\tau, S_\tau)$.

- The sample code does not contain the hedging problem. This is left as an exercise.

## Hedging results

Hedging errors $\hat{V} + \sum_{m=0}^{\tau^{\theta}M-1} h^{\lambda_m}(\tilde{w}_m^k) \cdot \big(p_{m+1}(\tilde{w}_{m+1}^k) - p_m(\tilde{w}_m^k)\big) - \tilde{g}^k$
for different numbers $MN$ of rebalancing times $(N = 3)$



## Some ideas how the code can be modified ...

1. Change the volatility from 20% to 40%. *How does the price change?*
2. Change the correlation between the stocks from 0% to 30%. *How does the price change?*

   *Correlated Brownian motions with correlation $c > 0$, can be generated as*

   $$V_t^i = \sqrt{c}W_t + \sqrt{1-c}W_t^i, \quad i = 1, 2, \dots, 5,$$

   *where $W_t, W_t^1, \dots, W_t^5$ are independent standard Brownian motions*
3. Use a local volatility model for $S_t^i$, $i = 1, \dots, 5$. Consider prices of the form

   $$S_t^i = s_0^i + \int_0^t (r - \delta_i) S_s^i \, ds + \int_0^t \beta_i(s, S_s) \, dW_s^i$$

   for

   $$\beta_i(t, x) = 0.6 e^{-0.05\sqrt{t}} \left( 1.2 - e^{-0.1t - 0.001(e^{rt}x - s_0^i)^2} \right) x$$

   *Use an Euler scheme to simulate the model*
4. Try to hedge the option.