E1. Neuronal Network for News Popularity Prediction

Final Report

Artificial Intelligence

3rd year of Integrated Master in Informatics and Computing Engineering

Group Members:

Neven Miculinic – up201510059 – neven.miculinic@gmail.com

Ricardo Magalhães – up201502862 – up201502862@fe.up.pt

Thursday, May 26th, 2016

Index

1. **Goal** 3
2. **Specification** 4
3. **Development** 5
4. **Experiments** 6
5. **Conclusions** 9
6. **Improvements** 11
7. **Resources** 12

1. Goal

The goal of this project is to implement an artificial neural network for predicting news popularity, based on number of social network shares. The dataset is from two years of Mashable articles, summarizing a heterogeneous set of features. We should successfully train a multi-layer neural network, in order to get a model capable of predicting new articles social network number of shares.

2. Specification

In this project data set, we have 58 predictive attributes, where some of it are binary vectorization of categorical variables (e.g. day of the week, data channel, etc.). It contains 39797 data points.

The articles were published by Mashable (www.mashable.com) and their content as the rights to reproduce it belongs to them. Hence, this dataset does not share the original content but some statistics associated with it. The original content be publicly accessed and retrieved using the provided urls. It was acquired on January 8, 2015.

We are going to scale all input features to [0,1] range. Also, we are going to divide our data set into three parts: training set (70%), cross validation (15%) and test set (15%). Categorical variables, such as day of the week and news category are already one-hot encoded. For base line prediction, we are going to use k-nearest-neighbor regression from *sklearn* Python library. Our aim is to implement a neural network, which has a better score than base line prediction. The cost function we are going to use to evaluate our news prediction score is mean squared error on data set.

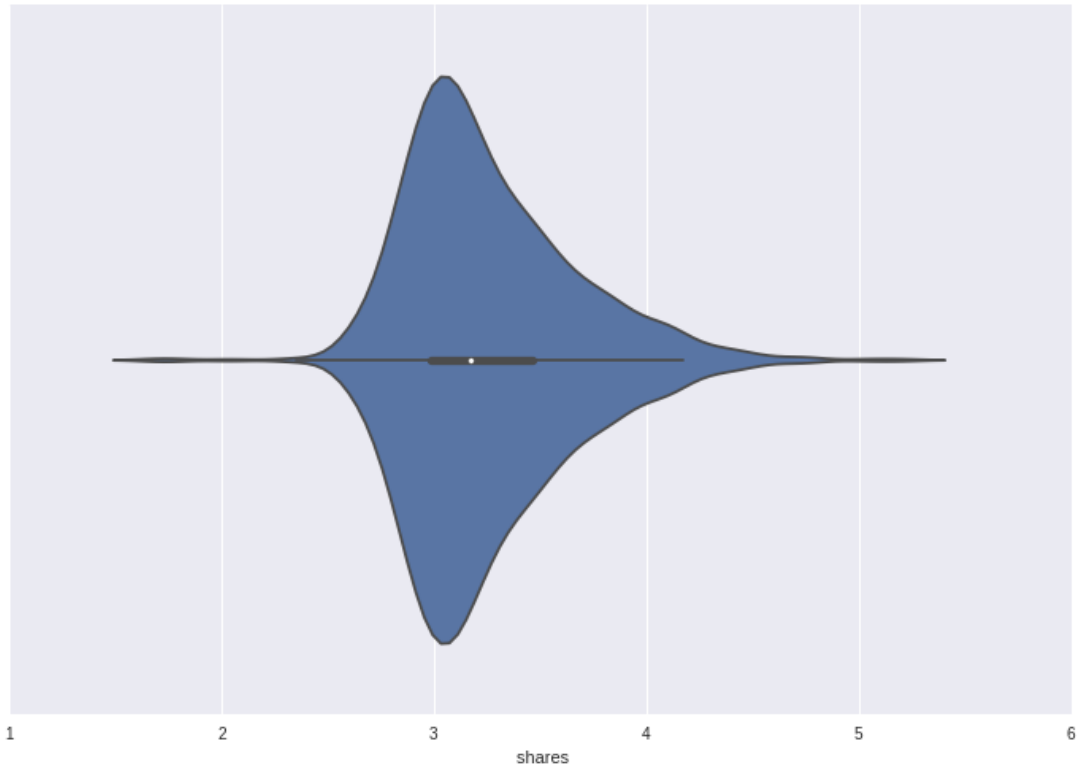

Figure 1 – Log_{10} share distribution

3. Development

In order to develop our neural network, we are using Python with *Jupyter Notebook*, previously known as *IPython*, to ease our work. There are a number of libraries used to help our development, such as:

1. ***TensorFlow*** is a open-source software library for numerical computation using data flow graphs; we are using it for all our computation.
2. ***Sklearn*** is machine-learning library for Python; we use it preprocessing the data and using off the shelf k-NN regressor.
3. ***TFLearn*** is high-level *TensorFlow* API; we are using it to simplify our *TensorFlow* graph construction.
4. ***NumPy*** is a Python package for scientific computing.
5. ***Pandas*** is a open-source library for high-perfomance and easy-to-use data strucutres; we are using it for data loading and plotting.
6. ***Seaborn*** is a data visualization library; we are using it for plotting.

We used five nearest-neighbors to interpolate the prediction.

4. Experiments

We first started to experiment with a linear regression learning algorithm to see how well it would perform. However, it turned out to be a bad predictor.

Then, we analyzed the correlation between the given dataset attributes and concluded there are some correlations between variables. After that, we used t-SNE to embed data in two dimensions in order to visualize it, as you can see in the following figure. T-SNE is non-parametric data dimensionality reduction, which preserves local structures.

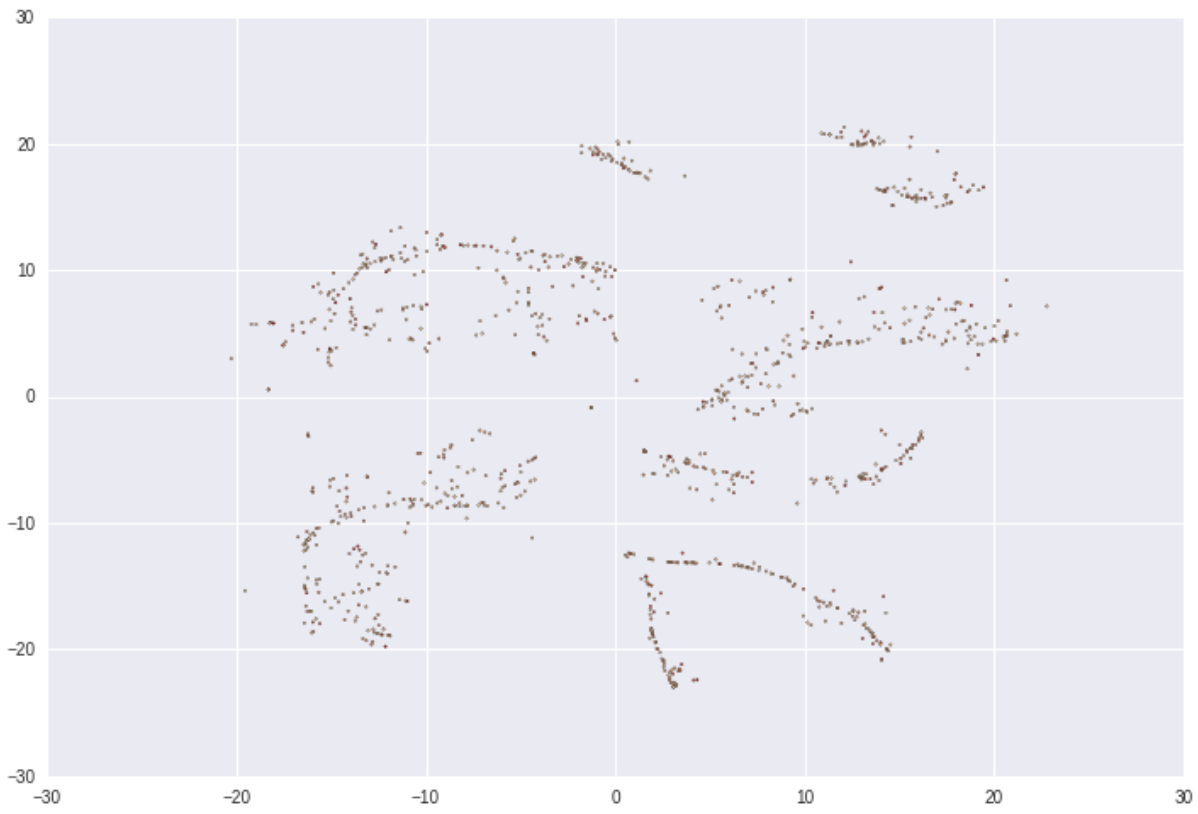

Figure 1 *– t-SNE Data Visualization*

With K-Means clustering, we divided our data into eight clusters and plotted the distribution of shares per cluster to see if there is any local structure we could introduce in our regression. However, as you can see in the following figure, the log distribution of shares remains constant per cluster.

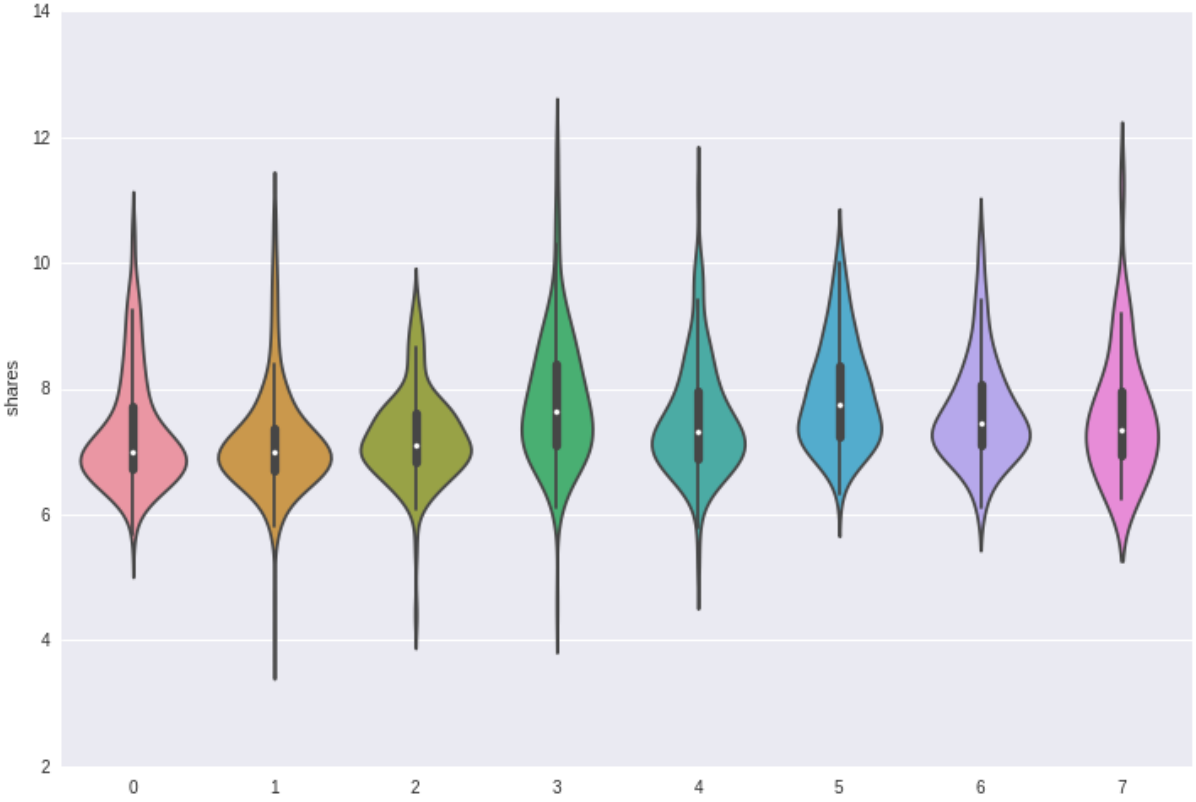

Figure 2 *– Log share distribution per cluster*

Our first approach was Mini-Batch Stochastic Gradient Descent with a batch size of 64, an algorithm used to minimize an objective function written as a sum of differentiable functions. We applied it with exponential decay learning rate and with varied network architectures (1-3 hidden layers with 10-100-1000 *relu*[1] neurons), in order to optimize and get better results. However, we applied Adam Optimizer[2] and Batch Normalization[3] techniques to speed up our trainings. Also, for further generalization, we tried L2 regularization with various penalties on weights, but it didn't seem to change the model much.

[1] "Efficient Backdrop" paper, 1998; Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Müller; *http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf*

[2] "Adam: A Method for Stochastic Optimization" paper, last revised on July 2015; Diederik Kingma, Jimmy Ba; *http://arxiv.org/abs/1412.6980*

[3] "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" paper, last revised on March 2015; Sergey Ioffe, Christian Szegedy; *http://arxiv.org/abs/1502.03167*

The model itself proved to be hard to over fit, because of small number of features, which behave stochastically on the number of shares.

5. Conclusions

In conclusion, we didn't find much variation between our experiments. L2 regularization didn't seem to influence much on the model learnt. Therefore, we trained simple model with 10,5 batch normalized relu neurons with Adam Optimizer, since this is a rather simple model and offers comparable performance against k-nearest-neighbor.

Batch normalization doesn't change our model capacity, but we noticed training is somewhat quicker with it; same goes for Adam Optimizer compared to previous approaches. Furthermore, as per paper[4] batch normalization has allowed us to greater learning rate and exhibits useful generalization properties.

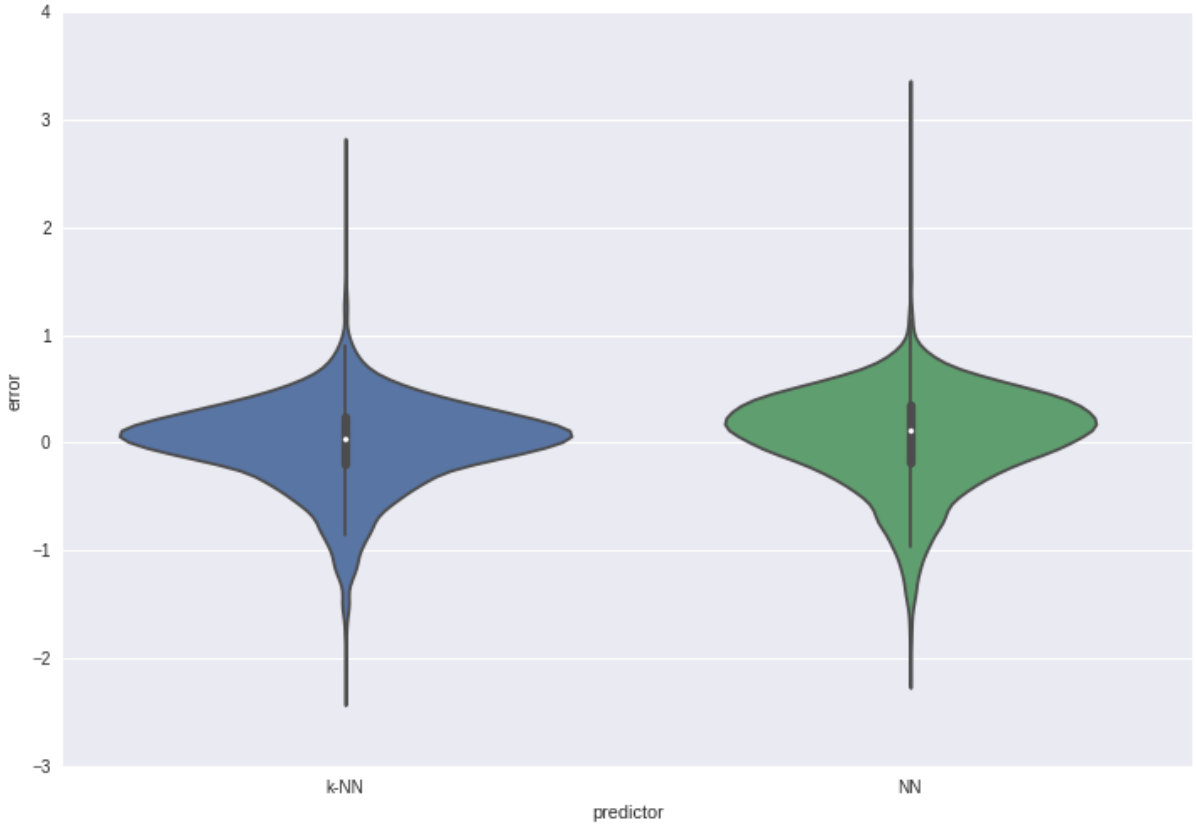

Figure 3 *– Relative prediction error on test set*

We conclude that data collected is insufficient for better prediction and that we either need more data or more features collected to perform better prediction. There is underlying stochastic data properties that make all our predictions err in both k-NN and neural

[4] "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" paper, last revised on March 2015; Sergey Ioffe, Christian Szegedy; *http://arxiv.org/abs/1502.03167*

network models similarity. Furthermore, various network architecture models exhibits similar error distribution, which supports this claim.

6. Improvements

To better improve our predictions, as said in the Conclusion, we either need more data and/or features. However, the only experiment with *relu* activation function and other activation functions could yield better performances.

7. Resources

1. "Adam: A Method for Stochastic Optimization" paper, last revised on July 2015; Diederik Kingma, Jimmy Ba; *http://arxiv.org/abs/1412.6980*
2. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" paper, last revised on March 2015; Sergey Ioffe, Christian Szegedy; *http://arxiv.org/abs/1502.03167*
3. "Efficient Backdrop" paper, 1998; Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Müller; *http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf*
4. "Artificial Intelligence: A Modern Approach" book, 3rd edition, December 2009; Peter Norvig, Stuart Russell
5. "Neural Networks and Deep Learning" web page, January 2016; Michael Nielsen; *http://neuralnetworksanddeeplearning.com/*
6. "Visualizing High-Dimensional Data Using t-SNE" paper, November 2008; L.J.P. van der Maaten and G.E. Hinton; *https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf*