Machine learning project

Leonardo Mirandola[1]

1. Aix-Marseille University, Marseille, France

Introduction

Artificial Intelligence (AI) popularity keeps on increasing as its knowledge is getting more accessible and easier tools to work with for Machine Learning (ML) and Deep Learning (DL) which are subsections of AI. Various models of ML and DL exists, such linear regression, logistic regression, Support Vector Machines (SVM), decision trees and neural networks; all used to predict either a continuous or categorical value. Each of these models have different complexity in its deployment and application based on the task. The main difference between ML and DL is the type learning, ML is supervised while DL is unsupervised, meaning the result of the dataset is not known, performing therefore in the clustering domain.

For this machine learning project, we used the data from the University of Irvine (UCI) with its machine learning repository data. The section we used is named "gene expression cancer RNA-Seq Data Set" which has the genetic expression of 20,531 genes for 801 patients who have one of five different cancers[1]. The ML model will therefore be a supervised training models predicting a particular cancer based on certain genetic expression from ground truth patients.

Therefore, this project will start by explaining the data processing section, then, we'll explore two different models used, neural networks, and decision trees. Finally, we'll see how each of these models performed.

Materials and methods

We used Python[2] 3.8.12 with TensorFlow[3] 2.7.0, Keras[4] 2.7.0, SciKit Learn[5] 1.0.2, XGBoost[6] 1.5.0, pandas[7] 1.3.5, NumPy[8] 1.19.5, and JupyterLab[9] 3.2.4.

The code was executed on a MacOSX 12.1, 16Gb RAM, M1 apple silicon, 1 Terabytes of storage. The notebook used for this project is available on github[1] or google colab[2]. The models presented are also available as unique output files to be launched for quick prediction.

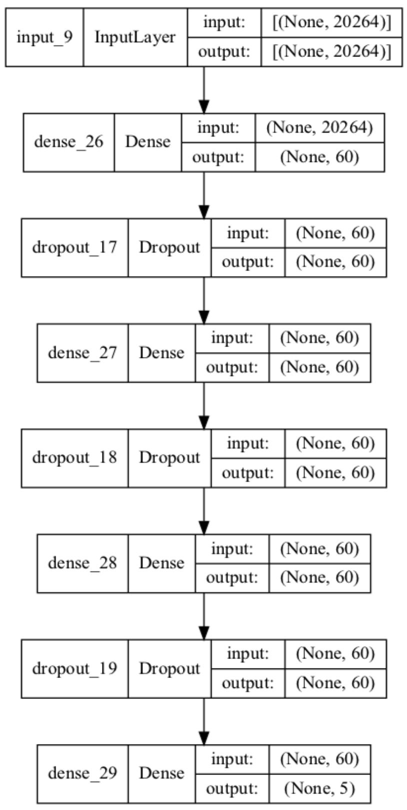
[1] https://github.com/LeoMira-1999/Machine-Learning-Project-Cancer
[2] shorturl.at/tBHXZ, click on open with google colaboratory after hitting link, results vary from colab and local machine

Every model is optimised to use multiprocessing or multithreading when possible with max number of jobs available to optimise training speeds.

The data collected was assembled to create a unified feature and label dataframe. Before creating the training and testing sets we checked to see if the data had columns with null values. After scanning, we find out that 267 columns (genes) have null values, thereby removing them from our set. Finally, we mixed and split the data into training and testing sets with a 30% testing and 70% training ratio.

Neural networks

Our model hyperparameters were defined using grid search from SciKit Learn to find a local best: epoch, batch size, layers amount and neurons amount per layer. Other hyperparameters weren't looked at because they wouldn't matter as much or were already optimal (activation, loss, initialization, optimizer) and would only complexify the search limiting our research time. Each run of the grid search had a cross-validation K-Fold of 5 meaning the data is split in 5 parts and bootstrapped to reduce order driven results.


input_9 | InputLayer | input: | [(None, 20264)] | output: | [(None, 20264)]
dense_26 | Dense | input: | (None, 20264) | output: | (None, 60)
dropout_17 | Dropout | input: | (None, 60) | output: | (None, 60)
dense_27 | Dense | input: | (None, 60) | output: | (None, 60)
dropout_18 | Dropout | input: | (None, 60) | output: | (None, 60)
dense_28 | Dense | input: | (None, 60) | output: | (None, 60)
dropout_19 | Dropout | input: | (None, 60) | output: | (None, 60)
dense_29 | Dense | input: | (None, 60) | output: | (None, 5)
Figure 1: Representation of the neural network architecture

The optimal neural network developed from grid search uses 8 layers, consisting of an input layer of shape 20,264, three dense middle layers of 60 neurons each using rectified linear unit (ReLU) activation function, best adapted for fitting our continuous data. In between the dense middle layers are two dropout layers which role is to drop 10% of the weights helping prevent overfitting by forcing the network to change its usual learning path and innovate. The output layer is 5 neurons networks allowing the classification using SoftMax activation function of the 5 different cancers, totaling 1,223,525 parameters.

The initialization of the network is done with random uniform and the network is built using the in-built Keras package inside of TensorFlow. The optimizer used is Adam for sparse data with categorical cross-entropy as the loss function adapted to our multi label class prediction which will help for backpropagation. The data had to be processed to be used by the model, this was done using NumPy tool "to_categorical".

Decision trees

Grid search was also used to optimize locally our hyperparameters: number of estimators, max depth, and learning rate, the grid search also uses cross-validation with a K-Fold of 3. The decision tree model of figure 2 is built using the XGBoost classification using the “multi:softprob” activation function for our multi label class prediction with the default gbtree booster. The objective function used as evaluation metric is mean log loss (mlogloss) being much faster at calculating multi class model metric than mean error (merror) model metric by default. XGBoost uses in-built gradient boosting (eXtreme Gradient Boosting) which combines weaker tree models to provide a better prediction, being a sub section of the general decision tree category. The data had to be processed before being used by the model in a different way than the other model using SciKit Learn tool LabelEncoder.

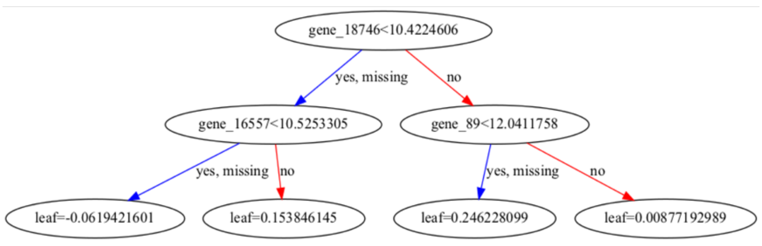


Figure 2: Representation of the decision tree using gradient boosting

Results

Neural networks

The results yielded by our grid search indicate that the best hyperparameters to use is 50, however 25 epochs work equally as good and is more time efficient, 3 layers and 60 neurons per layer. Therefore, on figure 3 we can visualize the learning curves for each epoch between accuracy and loss for training and testing data. We see that within the first 8 epochs the model reaches near 1 for accuracy and near 0 for loss, both training and testing. The transparent spikes we see after the epoch 10 might be due to dropout randomly dropping an important prediction feature causing a lowering in accuracy and increase in loss. We can also see that after more than 20 epochs the model stabilizes and plateaus.

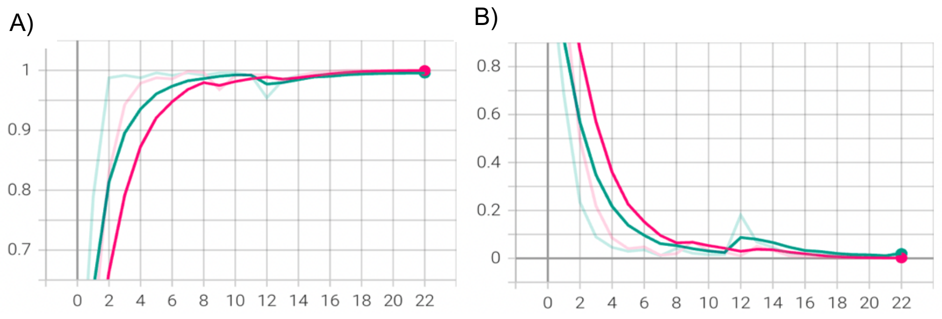


Figure 3: A) Measurement of the accuracy (y-axis) over epochs (x-axis) for train (pink) and validation data (light green). B) Measurement of the loss (y-axis) over epochs (x-axis) for train (pink) and validation data (light green). For both graphs a smoothness of 0.6 has been applied, in light transparent are the original curves. Y-axis maximizes to epoch 22, being an in-built graph limitation step size of 2, but model reached 25 epochs.

col_0	**0**	**1**	**2**	**3**	**4**
row_0					
0	41	0	0	0	0
1	0	41	0	0	0
2	0	1	90	0	0
3	0	0	0	44	0
4	0	0	0	0	24

Figure 4: Classification of predictions from testing set, on the diagonal are the rightfully predicted.

The model isn't showing any signs of overfitting as we would expect from an accuracy of 99.5% and loss of less than 0.01, first indicated by the light green validation data from figure 3 and from figure 4 showing that only 1 out of all the testing data has been wrongly classified.

Decision trees

The grid search used for the gradient boosting revealed the best hyperparameters were learning rate of 0.1, max depth of 3 and number of estimators being 100. The resulting accuracy of this model is of 99.17% on testing data and 99.1% on training data, thereby showing no overfitting on the training data.

Conclusion

To conclude with, we found out that both neural network and decision trees using gradient boosting allows for excellent prediction accuracy with no overfitting. In terms of accuracy neural networks takes the lead by a very small margin of 0.4%, however, being a more complex model, it will run slower than the gradient boosting model when classifying large batches of data.

Bibliography

1. UCI Machine Learning Repository: gene expression cancer RNA-Seq Data Set. Accessed January 8, 2022. https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq

2. Van Rossum G, Drake FL. *Python 3 Reference Manual*. CreateSpace; 2009.

3. Martín Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Published online 2015. https://www.tensorflow.org/

4. Chollet F, others. Keras. Published 2015. https://github.com/fchollet/keras

5. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python. *J Mach Learn Res*. 2011;12(Oct):2825-2830.

6. Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM; 2016:785-794. doi:10.1145/2939672.2939785

7. McKinney W, others. Data structures for statistical computing in python. In: *Proceedings of the 9th Python in Science Conference*. Vol 445. Austin, TX; 2010:51-56.

8. Harris CR, Millman KJ, Walt SJ van der, et al. Array programming with NumPy. *Nature*. 2020;585(7825):357-362. doi:10.1038/s41586-020-2649-2

9. Kluyver T, Ragan-Kelley B, Pérez F, et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. In: Loizides F, Schmidt B, eds. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press; 2016:87-90.