Is this the real life or just fantasy?

Alexandre Santos[1 [2018277730]] and Nuno Moita[2 [2018284367]]

University of Coimbra, Coimbra, Portugal
https://www.uc.pt/fctuc/
{uc2018277730,uc2018284367}@student.uc.pt

Abstract. In this paper we access models that are able to perform image classification with the objective of fraud detection. Through a established pipeline we performed operations with the data in order to compare the algorithm performances and decide which one is best suited for the situation. Concepts such as Ensembles, Support Vector Machines, Artificial Neural Networks will be explained and explored.

Keywords: Ensembles · Support Vector Machines · Convolution Neural Networks.

1 Problem Description

Can you tell what is real from what is fake? In a world where machine models are capable of creating realistic content starting from pure noise, models that can create images, text, audio and videos from scratch with the just the proper inputs, how sure can you be that the content you are experiencing isn't just a machine's fabrication? These are times where we need to become more vigilant of what we see on the Internet and where we need to rely on trusted news sources. How we move forward in the age of information may be the difference between whether we survive or end up in some mind bending dystopia. Luckily for us, **Machine Learning (ML)** enables us to develop models that are able to tell if such content is either fabrication or if is true.

In this work, we are going to embrace this theme by creating models that are able to solve this problem and for this we will use the datasets provided by a *Kaggle* competition. In the end we will use the model with best scores to classify the images and submit a **CSV** file with the **id** of the image and the corresponding **label**.

2 Approach

The dataset given has 40000 images to train and test the algorithms, from this set 20000 images are real and the other 20000 are fake. And finally there's 2000 images to classify for the submission competition. Each image has a size of **28x28 pixels** and it's color mode is **greyscale**. In image classification problems we have

to take into account the number of pixels, image size, color mode, number of channels and others. All these attributes affect how the data preparation process is defined and done.

2.1 Data Preparation

First we loaded the data into two numpy arrays in order to facilitate the model training, one corresponding to the data of all images and other corresponding to the labels. When the corresponding label to the image is 0 it means it's **fake**, if it's 1 it means is **real**. After we **normalized** the data and since we are dealing with a highly dimensional dataset, having each image **784 features**(28*28 pixels), we apply **Principal Component Analysis (PCA)** as the linear dimension reduction technique. Through this method we project our original dataset into 200 principle components having a sum of **95.38% explained variance** so we can speed up the machine learning algorithm's training and testing time.

2.2 Data Distribution

After the dimensionality reduction we used **Train-Validation-Test(TVT)** split on the data to use for the model prediction analysis and this were the resulting sets:

- Train Set (60%)
- Validation Set(20%)
- Test Set(20%)

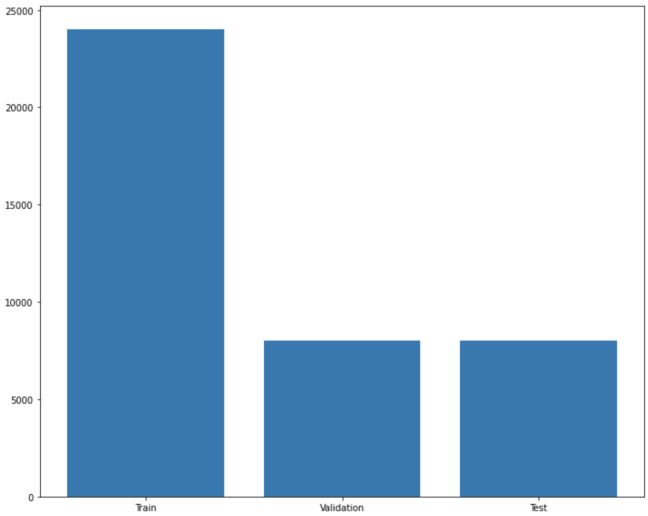

Fig. 1: Data Distribution after the TVT

After these operations on the data we have the tools to begin the training, parameter testing, analyse and compare the results to choose the best model for the problem at hand.

3 Model Prediction Analysis

The problem is image classification, so the algorithms we chose were with that in mind, so we'll be analysing:

- Support Vector Machines
- Artificial Neural Networks
- Ensembles

In order to select the best model out of the algorithms, we must go through a deep analysis of the parameters of each one. This process is known as **Hyper-parameter Tuning**. During this process, the validation results obtained after testing of different parameters in multiple models are compared based on their **accuracy**, **precision**, **recall** and **AUC**. It is also important to note that due to it being the most relevant, accuracy has more weight compared to the other values as of deciding the optimum parameter.

3.1 Support Vector Machine (SVM)

SVM's are algorithms commonly used for supervised machine learning models. This algorithm separates data into different classes of data by using a hyperplane, supported by the use of a support vector.

Kernel Function

A kernel is a function used in SVM for helping to solve problems. They provide shortcuts to avoid complex calculations. Choosing the right kernel is crucial, because if the transformation is incorrect, then the model can have very poor results. After testing both **linear** (SVM1), **polynomial** (SVM2), **radial basis function (RBF)** (SVM3) and **sigmoid kernels** (SVM4).

	accuracy	**precision**	**recall**	**roc_auc**
SVM1	0.76775	0.762443	0.768626	0.767761
SVM2	0.948625	0.951986	0.945096	0.948631
SVM3	**0.95575**	**0.955757**	**0.955276**	**0.955748**
SVM4	0.549375	0.543741	0.547179	0.549349

Table 1: Validation results of SVM models with different kernel

The results on Table 1 show that the best parameter is **RBF** having the best metrics score on all categories.

Regularization parameter (C)

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. For choosing C we tested commonly used values such as: **0.01**(SVM5), **0.1**(SVM6), **10**(SVM7).

	accuracy	**precision**	**recall**	**roc_auc**
SVM5	0.651875	0.577174	0.675493	0.654960
SVM6	0.8655	0.876320	0.856511	0.865660
SVM7	**0.976750**	**0.975113**	**0.978064**	**0.976761**

Table 2: Validation results of SVM models with different C

Analysis of the validation results showed that the most fitting value is **10**.

Gamma

Gamma is only used when we use the Gaussian RBF kernel, if you use linear or polynomial kernel then you only need C. This parameter decides how much curvature we want in a decision boundary. The values utilized for Gamma are similar to the ones used for the C parameter. The values tested were: **0.01** (SVM8), **1**(SVM9), **10**(SVM10).

	accuracy	**precision**	**recall**	**roc_auc**
SVM8	**0.972625**	0.970337	**0.974501**	**0.972643**
SVM9	0.61075	0.993213	0.561381	0.766657
SVM10	0.497375	**1.0**	0.497312	0.748656

Table 3: Validation results of SVM models with different gamma value

Testing concluded that the model performed best with Gamma values set to **0.01**.

Best Model After analysing the previous parameters we came to the conclusion that **SVM7** is the best model from the algorithm at question having the following parameters:

- Kernel Function: **RBF**
- C: **10**
- Gamma: **scale** (default in *sklearn* library)

3.2 Artificial Neural Networks (MLP)

The architecture known as Multi Layer Perceptron (MLP) consists of 3 layers: **Input**, **Hidden** and **Output** where groups of multiple perceptrons/neurons are located. MLP Classifiers rely on this underlying Neural Network to perform the task of classification.

Activation Function

The Activation Function in the hidden layer will control how well the network model learns the training dataset. And in the output layer will define the type of predictions the model can make.

For this purpose we considered both **Identity**(MLP1), **Logistic (Sigmoid)**(MLP2), **Hyperbolic Tangent (Tanh)**(MLP3), **Rectified Linear Activation (ReLU)**(MLP4) as parameters of our Hidden Layer Activation Function. As of before, the results of the parameter comparison determine which is to be used, in this case, ReLU.

	accuracy	**precision**	**recall**	**roc_auc**
MLP1	0.765625	0.764455	0.764262	0.765618
MLP2	0.966375	0.961538	0.970566	0.966436
MLP3	0.962375	0.956762	0.967217	0.962452
MLP4	**0.970250**	**0.963298**	**0.976555**	**0.970367**

Table 4: Validation results of MLP models with different activation function

Solvers

The solver serves as an optimization algorithm and is a crucial part of the learning process. We tested two of the most common first order optimization algorithms: **Stochastic Gradient Descent (SGD)** and **ADAM**. These algorithms were then compared to another algorithm of second order: **Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)** .

	accuracy	**precision**	**recall**	**roc_auc**
MLP5	0.958875	0.952489	0.964368	0.958971
MLP6	0.95075	0.947964	0.9527539	0.950771
MLP7	**0.969125**	**0.964555**	**0.973117**	**0.969181**

Table 5: Validation results of MLP models with different solver

After testing, the results showed that the **ADAM** algorithm was the one that best fitted our objective function.

Learning Rate

The learning rate controls how much to change the model in response to the estimated error each time the model weights are updated. A smaller value may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. The tests were made using **Constant** (MLP8), **Inverted Scaling**(MLP9) and **Adaptive**(MLP10) learning rates.

	accuracy	**precision**	**recall**	**roc_auc**
MLP8	**0.973125**	**0.968326**	**0.977417**	**0.973187**
MLP9	0.969875	0.962544	0.976537	0.970004
MLP10	0.971625	0.966315	0.976378	0.971697

Table 6: Validation results of MLP models with different learning rate

After analysing the validation results, we concluded that our best option was to keep the a Constant Learning Rate.

Hidden Layer Size

Since increasing the size of a hidden layer is less expensive, in computational terms, than increasing the number of hidden layers this value needs to be closely adjusted to the complexity of the problem. We tested for different amounts of nodes in the two hidden layers: $(150, 3), (200, 4), (250, 5), (300, 5), (350, 5)$, respectively as seen in Table 7.

	accuracy	**precision**	**recall**	**roc_auc**
MLP11	0.970875	0.967320	0.973931	0.970912
MLP12	0.958	0.945701	0.969088	0.958318
MLP13	0.972875	0.967823	0.977405	0.972943
MLP14	0.973125	**0.970588**	0.975246	0.973147
MLP15	**0.97375**	0.966817	**0.980122**	**0.973869**

Table 7: Validation results of MLP models with different size of hidden layers

As we can see the more nodes in a hidden layer the better the model is in general, expect in the precision section where the model MLP14 has a better result than the MLP15.

Best Model

After analysing the validation results for each parameter we select the best ones and put together to form the final MLP model, and after analysing the best model is **MLP15** with following parameters:

- Activation Function: ReLU
- Learning Rate: Constant
- Solvers: ADAM
- Hidden Layer Size: (350,5)

3.3 Convulotionary Neural Networks (CNN)

CNN models are at their core composed by various filters, that are used to extract the relevant features from the input. These Neural Networks generate filters automatically without mentioning it explicitly. These filters help in extracting the right and relevant features from the input data. It attends to the spatial features from an image, which by itself refers to the arrangement of pixels and the relationship between them in an image. Also follows the concept of parameter sharing. A single filter is applied across different parts of an input to produce a feature map.

Architecture

As opposed to previous models mentioned in this paper, no PCA was defined as of processing the dataset utilized by the CNN. The architecture we used for our CNN was based on the LeNet architecture. It is composed of convolutional layers, followed by max pooling in between the convolutional and the upcoming dense layers.

The original **LeNet** architecture used TANH activation functions rather than ReLU . The reason we use RELU in the activation function of most layers is because it tends to give much better classification accuracy. A logistic activation function is also present in the last "one-neuroned" dense layer.

	accuracy	**precision**	**recall**	**roc_auc**
CNN1	0.0.776750	0.849625	0.754875	0.795875
CNN2	0.554835	0.700150	0.511216	0.592971
CNN3	**1**	**1**	**1**	**1**
CNN4	0.845341	0.884126	0.835182	0.854741

Table 8: Validation results of CNN models

After testing for different filter numbers and varying the dropout values we can say for sure the **CNN3** model with 32 filters and a dropout rate of 0.6 is the preferable model.

3.4 Ensembles

It's a machine learning technique that combines several base models in order to produce one optimal predictive model.

The three main classes of ensemble learning methods are:

- Voting
- Bagging
- Boosting

Inside these lie a combination of the Logistic Regression, K-Nearest Neighbor Voting, Decision Tree Classifier, Gaussian Naive Bayes, MLP and SVM models.

Voting

A voting ensemble is an ensemble machine learning model that combines the predictions from multiple other models, ideally achieving better performance than any single model used in the ensemble.

Majority Voting takes place when the predictions for each label are summed so the label with the majority vote is decided. Nonetheless, a limitation of the voting ensemble is that it treats all models the same, meaning all models contribute equally to the prediction. This can be a problem if some models are good in some situations and poor in others.

Majority Weighted Voting consists on a weighted ensemble and is an extension of a model averaging ensemble where the contribution of each member to the final prediction is weighted by the performance of the model. We cross-validated each algorithm that belongs to the ensemble and gave them weights depending on their performance, for example the SVM was the model with best accuracy score so it has more weight then others and so on.

	accuracy	**precision**	**recall**	**roc_auc**
Majority Voting	0.937125	0.932127	0.940878	0.937180
Majority Weighted Voting	**0.95375**	**0.946455**	**0.959969**	**0.953869**

Table 9: Validation results of Voting models with different mechanisms

These two voting ensembles were then compared and the validation results pointed towards the use of the Majority Weighted Voting Ensemble.

No doubt that **Majority Weighted Voting** ensemble technique has the best results therefore the better model.

Bagging

Bootstrap aggregation, or bagging for short, is an ensemble learning method that seeks a diverse group of ensemble members by varying the training data.

A Bagging classifier fits base classifiers each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction

Random Forest is an extension of bagging, a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Number of estimators

In general, the more trees are in the model the better the results. However, the improvement decreases as the number of trees increases, and eventually the benefit in prediction performance from learning more trees will be lower than the cost in computation time for learning these additional trees.

The validation results for the values, by the order: **100**(RF1), **200**(RF2), **300**(RF3), **400**(RF4), **500**(RF5) for the number estimators are shown in Table 10.

	accuracy	**precision**	**recall**	**roc_auc**
RF1	0.873375	0.889643	0.860442	0.873751
RF2	0.89625	0.924334	0.8742272	0.897446
RF3	0.903125	0.934892	0.878158	0.904683
RF4	0.90625	0.939165	0.880301	0.907936
RF5	**0.910875**	**0.944947**	**0.883847**	**0.912690**

Table 10: Validation results of RF models with different number of trees

We were able to conclude from the results that the model achieved improved performance with 500 estimators.

Criterion

The function to measure the quality of a split.

For this we tested both for the **Gini Index**(RF6), which measures the frequency at which any element of the dataset will be mislabelled when it is randomly labeled and **Entropy**(RF7), that is a measure of information indicating the disorder of the features with the target.

	accuracy	precision	recall	roc_auc
RF6	0.90675	**0.940170**	0.880414	0.908491
RF7	**0.908**	0.939668	**0.882853**	**0.909562**

Table 11: Validation results of RF models with different criterion

The latter seemed to provide better results from testing, so we keep it.

Max Features

Increasing max features generally improves the performance of the model as at each node now we have a higher number of options to be considered. Hence, we need to strike the right balance and choose the optimal max features. We tried calculating the amount of max features through the use of a **Square Root Function**(RF8), a **Logarithmic Function with base 2**(RF9) and by simply assigning the **default**(RF10) amount of features.

	accuracy	precision	recall	roc_auc
RF8	**0.910875**	**0.943690**	**0.884751**	**0.912565**
RF9	0.904875	0.944193	0.874505	0.907293
RF10	0.896125	0.922826	0.875089	0.897203

Table 12: Validation results of RF models with different number of max features

From the values represented in this table, we decide to keep the **Square Root Function** as it shows the best results.

Best Model So after carefully reviewing the metrics scores of each model, the best one is either **RF5** or **RF8** since they are using the same option to define the max features.

- Number of estimators: 500
- Criterion: Entropy
- Max features: Square Root Function

4 Best Model Selection

Since we now have all models selected with the most efficient parameters we now run the testing process one last time.

	accuracy	**precision**	**recall**	**roc_auc**
SVM7	0.97675	0.975113	0.978064	0.976761
MLP15	0.97375	0.966817	0.980122	0.973869
Majority Weighted Voting	0.95375	0.946455	0.959969	0.953869
RF8	0.910875	0.94369	0.884751	0.912565
CNN3	**0.8525**	**0.7084259945638745**	**1.0**	**0.8850574712643678**

Table 13: Validation results of the best models

Based on these results of the test set we decide to choose the CNN3 model to classify the images of the competition or other situations where this tool might seem adequate.

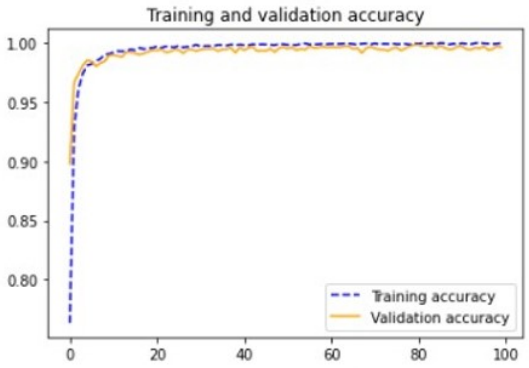

Fig. 2: Training and Validation Accuracy

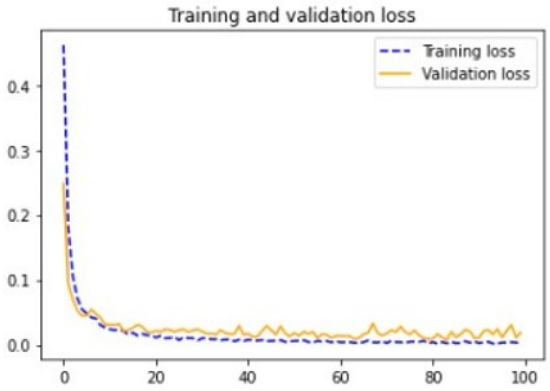

Fig. 3: Training and Validation Loss

Further analysis of the data provided in Fig.[2,3], concludes that the learning curve originated by the CNN3 model came out a good fit, neither over or under fitting. Both the Training loss and Validation loss are close to each other with validation loss being slightly greater than the training loss. Another characteristic of a good fit is the initially decreasing training and validation loss and a flat training and validation loss after some point till the end.

5 Conclusion

Lastly, we uploaded the final dataset to *Kaggle* competition, where we obtained a 0.50700 score. We were lucky to have a dataset with small images and with the color mode set as greyscale, it greatly contributed to the model's training speed, giving us more time to explore distinct algorithms for the selection of the model through the competition. Unfortunately we missed some of the algorithms i.e. the **Gradient Descent Boosting** as a ensemble boosting technique and **Recurrent Neural Networks (RNN)**. Having learned the concepts, implemented the algorithms and analysed the results it's fair to say that the CNN is the most efficient machine learning model for image classification problems, at least from the models that we analysed. The process of parameter tuning of this model has a lot to offer since it offers the possibility of having different architectures and having much more variables to play with.

References

1. ScikitLearn, https://scikit-learn.org/. Last accessed 21 May 2022
2. Medium, https://medium.com/@myselfaman12345/c-and-gamma-in-svm-e6cee48626be. Last accessed 21 May 2022
3. Machine Learning Mastery, https://machinelearningmastery.com/. Last accessed 21 May 2022

4. PyImageSearch, https://pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/. Last accessed 21 May 2022
5. Analytics Vidhya, https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/. Last accessed 21 May 2022
6. QuantDare, https://quantdare.com/decision-trees-gini-vs-entropy/. Last accessed 21 May 2022
7. Kaggle, https://www.kaggle.com/questions-and-answers/196752. Last accessed 21 May 2022
8. Ensembles, https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f. Last accessed 21 May 2022
9. PCA, https://www.datacamp.com/tutorial/principal-component-analysis-in-python. Last accessed 21 May 2022
10. Towards Data Science, https://towardsdatascience.com/learning-curve-to-identify-overfitting-underfitting-problems-133177f38df5. Last accessed 22 May 2022