Authorship Identification from Artworks

Yueyang Zhang Shichao Liu Lunhui Chen Yutao Tu

April 27, 2020

Abstract

This project presents different methods for artworks classification problem with a dataset that contains 4299 images of artworks belonging to 11 different artists. We implement data preprocessing, and then fit several tree-based models, Support Vector Classifier and pretrained Neural Network (CNN) model Resnet50 to classify the images into 11 classes. We compare the performances of the models and finally achieved a test accuracy of 92.73% with the pretrained Resnet50 model.

1 Introduction

The increasing volume of digital artworks on the internet provides an opportunity for all individuals to immerse themselves in the art at any arbitrary place and time. However, unlike these artworks that archived in the museum collections, there are some online artworks usually without detailed information. The visual style is a significant break to identify the authorship. The definitions of such artworks are unclear since styles would overlap, or artists would have multiple styles that will cause difficulties in authorship recognition. Rather than the experts in arts, it is impossible for all people to learn all the visual styles to gain insights into the authorship from artworks. Therefore, we need to classify, analyze, and understand these unknown paintings in an automatic manner.

Advanced machine learning methods provide the opportunity to analyze these paintings. These algorithms could help to classify artists in the artwork by being more accurate than even trained art experts. For this project, we would like to present computational techniques based on machine learning algorithms for developing an automatic classifier to recognize artists. We will try different machine learning methods in order to improve the accuracy of our model. The final classifier presented in the study will allow people to easily understand these unknown artworks that promote them have new insights into the stylistic development of different.

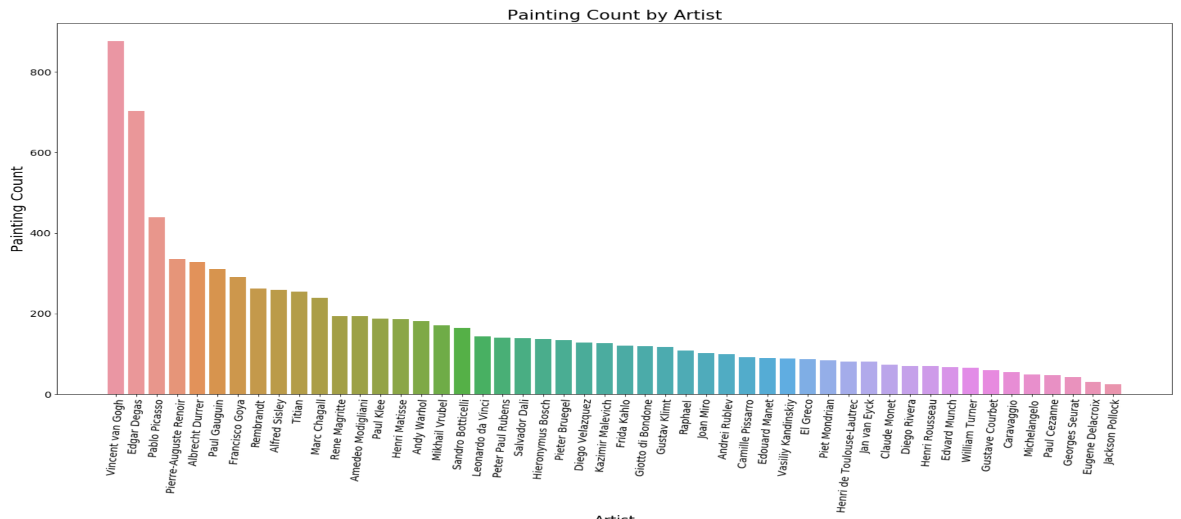


Figure 1: Painting Count by Artists

2 Data Description and Preprocessing

Our data are retrieved from a Kaggle Dataset: Best Artworks of All Time[1]. The original data set contains 8683 artworks from 50 Medieval European artists, and Figure 1 shows the total number of artworks for each of 50 artists.

To ensure our data have enough sample size for each artist, we will only focus on those artists who have at least 200 artworks contained in this data. After this filtering process, we select total 4299 artworks from 11 artists. Table 1 contains the number of artworks included for each artist. We also perform training-test split on our data. In particular, we first group the artworks by artists (total 11 groups), and among each group, we randomly split the data into training set and test set, and training set contains 70% of the observation in the original group, and test set contains the rest 30%. After doing this, we combine 11 training sets into a "larger" training set as our training data, and we do the similar combining process to obtain our test data.

In order to fit statistical models using these artworks, we have to convert raw images of these artworks to numerical values to make data practical. And since all images in our data have different sizes, we need to resize each of them to the same size in order to keep the consistency of data. Specifically, we first used `readImage` function in `OpenImageR` package to open each image in R and convert it into numerical data, and we used `resizeImage` function with bilinear scaling method to resize each image to 128×128 pixels. The reason to choose size 128×128 is because if resolutions of images are too high, the model training time would be incredibly long, or even would not complete. In fact, we had tried to use a higher resolution for converted images: 256×256, but due to technical limitations, our machine had crashed multiple times when converting resized images to numerical data, and we could not finish this conversion process. However, our machine could successfully complete this in several hours if we lowered the target

[1]https://www.kaggle.com/ikarus777/best-artworks-of-all-time

Name of artist	Number of artworks
Vincent van Gogh	877
Edgar Degas	702
Pablo Picasso	439
Pierre-Auguste Renoir	336
Albrecht Durer	328
Paul Gauguin	311
Francisco Goya	291
Rembrandt	262
Alfred Sisley	259
Titian	255
Marc Chagall	239

Table 1: Number of artworks for 11 selected artists

resolution down to 128×128. And if resolutions are too low, it might be hard for our models to distinguish different images from different artists, and hence yielding poor classification performance. Thus, 128×128 is the result of the trade-off between model training time and prediction precision.

After this conversion process, each artwork was converted into a 49152-column observation in our data. Each column represents a pixel with numerical value ranges from 0 to 1 (continuous), and each pixel is a explanatory variable (predictor) in our research. Our response variable is simple: the corresponding artist name of the specific artwork. Together, our finalized data set contains 4299 rows (3004 rows in training set, 1295 in test set) and 49153 columns (49152 predictors and 1 response), where each row represents a specific artwork. Since the number of predictors are very large (much larger than the number of observations), we performed Principal Component Analysis (a.k.a. PCA) to reduce the dimension of our data. Details of the implement of PCA are included in Section 3.1.1.

3 Methodology

3.1 Method Implement

3.1.1 Principal Component Analysis

The artwork image data we got have $128 \times 128 \times 3(\text{RGB}) = 49152$ dimensions. The high-dimension data is of heavy computational cost to implement classification methods, hence we first implement Principal Component Analysis to linearly reduce dimensions from 49152 to 150.

Let $p = 49152$, and we do linear transformation $Z = U^T X$

$$\begin{cases} Z_1 = u_{11}X_1 + u_{21}X_2 + \cdots + u_{p1}X_p, \\ Z_2 = u_{12}X_1 + u_{22}X_2 + \cdots + u_{p2}X_p, \\ \vdots \\ Z_p = u_{1p}X_1 + u_{2p}X_2 + \cdots + u_{pp}X_p, \end{cases} \tag{1}$$

and subject to $\|u_i\|_2 = 1$, $cov(Z_i, Z_j) = 0$, and $Var(Z_1) \geq Var(Z_2) \geq \cdots \geq Var(Z_p)$. We choose the first $k = 150$ principal component to reach a cumulative contribution rate $= \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i} = 80\%$. And then applied this dimension-reduced dataset for further model constructing.

3.1.2 Random Forest

Random Forest is an extension and an improvement of classification tree, which uses bootstrapped training set of data to fit multiple classification trees. And at each tree split, Random Forest randomly selected a smaller set of predictors at each split in a tree to make the correlations between trees as lower as possible. Since Random Forest would generally produce better classification result than a single classification tree, we implemented this method in our research.

We basically used `randomForest` function in `randomForest` package to fit Random Forest models. Specifically, instead of doing 10-fold cross-validation, which would take significant amount of time on our data, we used 5-fold cross-validation to select the tuning parameter `mtry`, which is the total number of variables used at each tree split. By using the selected `mtry` and the total number of trees fitted = 500, we eventually obtained the cross-validated accuracy rate of 51.76%.

We also used this trained model to make prediction over the test data, and its test accuracy is approximately 53.4%. This result is also included in Table 2.

3.1.3 XGBoost

Another tree-based classfier we used is XGBoost. XGBoost stands for eXtreme Gradient Boosting.It actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms.

We add one tree each step to fitting the error between our true y value and the fitted values based on the trees we have in the previous step. That is:

$$\begin{aligned} \hat{y}_i^{(0)} &= 0, \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i), \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i), \\ &\vdots \\ \hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \end{aligned} \tag{2}$$

The objective function is in the form of:

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned} \tag{3}$$

where $\Omega(f_i)$ is a regularization term. XGBoost use second order Tailor expansion to approximate the objective function and then the problem will be the same as the optimization problem for quadratic functions. The culmulative score of each of tree will be the final decision of XGBoost.

When implementing this method, we choose `xgb.XGBClassifier` in `xgboost` package in python, and use the learning rate = 0.5 and the minimum loss reduction required to make a further partition on a leaf node of the tree (gamma) to be 0.1 for training our model.

3.1.4 Support Vector Classifier

Support Vector Classifier is a gneralized linear classifier that produce hyperplanes that can seperate classes most. It still effective when the number of dimensions is greater than the number of samples. In Python, We use `SVC` function in `sklearn` package and use linear kernel to implement the modeling building.

3.1.5 Resnet50 CNN model

ResNet was first proposed by four Chinese, Kaiming He etc, who were in Microsoft Research Institute.[1] Deeper Convolutional neural networks are more difficult to train. A residual learning framework can ease the training of networks that are substantially deeper than those used previously. Resnet 50 is of 50 layers deep. The pretrained version of the network was trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories. As a result, the network has learned rich feature representations for a wide range of images.

For transfer learning, we first load the model with pre-trained weights which have been trained on ImageNet. Then, we freeze the weights in existing layers and replace the fully connected layer with a layer where the number of outputs matches the number of classes of our dataset, which is 11. Cross-entropy loss has been selected as the loss function and we use stochastic gradient descent in the training process. To compare with other methods, here we used images resized as 128×128 pixels. And the batch size is tuned as 64.

3.2 Results Comparison and Discussion

Table 2 includes the testing results of models we implemented before. By comparing the results across different models, we can see that the CNN pretrained model performs the best. The other three methods do not achieve a good performance. And we have

Methods	Test Accuracy
Random Forest after PCA	53.4%
XGBoost after PCA	52.0%
SVC after PCA	52.9%
CNN Resnet50	72.7%

Table 2: Models Results Comparison

checked that the PCA step in the three methods have little effect on the performance, but have significantly reduced training time.

As a result of model selection, we choose to use Resnet50 model as our final model. But it should be realized that this is due to multiple reasons including computer computing power limitation, data preprocessing methods, parameters tuning and so on. So we cannot simply say that one model is better than another. Even when using the same Resnet50 network and the same pretrained parameters, different methods of tuning learning rate would lead to different performances. In our case, we choose to use cyclical learning rates to train models quicker and with higher accuracy by `fit_one_cycle()` function. [2]

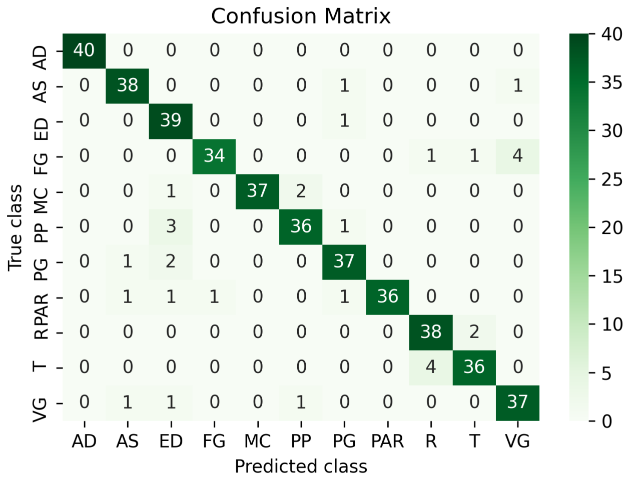
4 Final Model and Result

Since we choose CNN model as our final model, we should feed as much data as possible to the model. We resized all images to 256×256 pixels (directly resizing original images, not converting them to numerical data) and set batch size as 64. Image augmentation including randomly flipping, rotating, zooming, lighting and etc has been applied to all training images. And to have a better evaluation of the performance and avoid the influence of imbalance data, we randomly choose 40 images from each class to form a balanced test set.

Our final model achieves a test accuracy of 92.73%, and a macro F1 score of 92.77%. The confusion matrix is shown in Figure 2.

5 Conclusion

We used data with image size 128×128 to fit tree-based models, support vector classifier, and convolutional neural networks (a.k.a. CNN) model, and it turned out that CNN has the highest testing accuracy among those models, whereas other three models have the similar testing performance at around 52% to 53%. With the best testing performance, we then further tuned CNN model in Python and used higher resolution images (256×256, using raw images instead of numerical data) to refit the model, and we eventually obtained an accuracy of near 93%. This is a very desirable



	AD	AS	ED	FG	MC	PP	PG	PAR	R	T	VG
AD	40	0	0	0	0	0	0	0	0	0	0
AS	0	38	0	0	0	0	1	0	0	0	1
ED	0	0	39	0	0	0	1	0	0	0	0
FG	0	0	0	34	0	0	0	0	1	1	4
MC	0	0	1	0	37	2	0	0	0	0	0
PP	0	0	3	0	0	36	1	0	0	0	0
PG	0	1	2	0	0	0	37	0	0	0	0
PAR	0	1	1	1	0	0	1	36	0	0	0
R	0	0	0	0	0	0	0	0	38	2	0
T	0	0	0	0	0	0	0	0	4	36	0
VG	0	1	1	0	0	1	0	0	0	0	37

Figure 2: Confusion Matrix of Testing Results

result, and according to the confusion matrix in Figure 2, only 32 out of 440 images are incorrectly classified using the tuned CNN model.

6 Future Plans

Obviously, the test accuracy for tree-based models (Random Forest and XGBoost) and Support Vector Classifier are relatively low comparing with CNN. Hence, in our future analysis, we will focus on further tuning those models to see if they can achieve better testing performances. Also, to improve the testing performances, we can try some other dimensional reduction methods on our original data to reduce the number of columns again, and use these new reduced data to refit models we used before. Specifically, we can try some nonlinear dimensional reduction methods, such as kernel PCA with several different kernels, and we can compare the results with those models fitted with standard PCA data. Moreover, if time permits, we can try some other classification methods as well if there is any.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[2] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," 2017.