CSC4025Z: Artificial Intelligence

Mapping Famous Artworks to a Low Dimensional Manifold for Classification

Zach Wolpe

12 November 2020

Abstract

In this paper I compare different machine learning techniques to classify famous artworks. Artworks are representative of high dimensional, sparse, spatial-temporal dependent rich data, which may prove problematic when applying statistical machinery. We show that classical statistical methods can achieve results on par with the cutting edge.

Additionally, the transfer learning instance is indicative of the modular nature of modern deep learning frameworks: providing an illustration of the modularity of these frameworks, which allow one to plug & play with model components - an extremely desirable feature. The full implementation is available on my Github.

Art History

One ought to have some scope of the artists of interest to properly appreciate the dataset. Here's some brief background on the artists in our study.

Albrecht Durer

Born May 21, 1471 in Nuremberg, Germany - a free imperial city (independent state) within the Holy Roman Empire - Durer earned a reputation as a phenomenal artist in his twenties for his outstanding woodcut prints. His most famous engravings are those detailed in his three Meisterstiche (master prints): *Knight, Death & the Devil* (1513); *Saint Jerome in his Study* (1514) and *Melencolia I* (1514).

Dürer was a painter, printmaker & theorist of the German Renaissance. A member of the High Renaissance movement, Dürer was in close communication with & connected to the leading Italian artists of the era: Raphael, Giovanni Bellini & Leanardo da Vinci.

Figure 2: *Praying hands*, also known as *Study of the Hands of an Apostle*, a pen-and-ink drawing by Dürer.

Alfred Sisley

Though primarily an impressionist, Alfred Sisley's influence spanned over the impressionist, post-impressionist & realist art movements. Sisley was born in (1839) & lived most of his life in France. Famous for his consistent impressionist landscape style - most often painting landscapes *en plein air* (outdoors) - Sisley occasionally deviated to figure painting. Many of his paintings in a classical pale iconic shades of pinks, purples, dusty blues & creams. Sisley's colour palette of choice and power of expression intensified over his life.

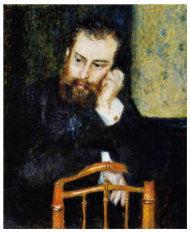

Figure 3: A portrait of Sisley painted by his good friend and colleague Pierre-Auguste Renoir - a seminal figure in the impressionist movement.

Rembrandt

One of the most famous & influential artists in history Rembrandt Harmenszoon van Rijn was a master draughtsman, painter & printmaker. Born 1606, Rembrandt is widely considered the most important artist is Dutch history - and undoubtedly one of the greatest visual artists of all time.

In contrast with the other Dutch masters of the 17^{th} century, Rembrandt produced works of a wide spectrum from self-portraits to genre and historical scenes, biblical and mythological themed studies, allegorical works and landscapes.

Figure 4: A self portrait of Rembrandt.

Dataset

Our dataset consists of 848 high resolution RGB images of paintings executed by our three artists, split:

- Albrecht Durer: 328
- Alfred Sisley : 259
- Rembrandt : 262

Our goal is to develop a multi-class classifier that is capable of discriminating between classes - in our case: *can the classifier predict which artist painted any given painting?*.

A nontrivial task given the dimensionality & spatial temporal structure of the data: assume each image is of size $(3, 320, 320)$ (most of which are larger) we effectively have $3 \times 320 \times 320 = 307'200$ parameters. Given our 848 samples clearly $n << p$ & as such we'll need to sufficiently extract features before any learning can take place.

1 Convolutional Neural Nets

ConvNets (Convolutional Neural Networks) are known to perform well on image classification tasks - having the ability to extract meaningful spatial temporal structure from the data.

One caveat in designing our model is the relatively small dataset size. Even an extremely simple neural network runs the risk of being greatly over-parameterized - able to copy the data & fail to generalize to unseen settings. To this aid we will utilized regularization, however, additional to this, a new body of literature suggests that the convolutional layers used for feature extraction detect similar features - proportional to layer depth - for ConvNets learned on vastly different datasets. That is to say that a the convolutional layers trained on any sufficiently complex image dataset can be used to extract meaningful features from another image dataset. This is the domain of transfer learning.

The convolutional layers primarily apply filters & down (up) sampling procedures to reduce the effective degrees of freedom in the data - improving the signal-to-noise ratio and extracting meaningful spatial invariant correlations across pixels to better represent the images in a computationally meaningful manner. Our dataset is very small and sufficiently complex (inconsistent across samples) that extracting the true underlying discriminatory relationships may prove non-trivial.

As such we utilized transfer learning to leverage pre-trained convolutional layers to perform meaningful feature extraction; & thereafter train a fully connected linear network to classify the features.

1.1 VGG16

We utilized the famous $VGG16$ network for this task. $VGG16$ was published in the paper *"Very Deep Convolutional Networks for Large-Scale Image Recognition"* in 2014. Originally trained on ImageNet (an open source large scale image dataset of over 15*million* high resolution colour images) $VGG16$ exceeded the current state of the art (AlexNet) by substituting large kernel-sized filters with multiple 3×3 stacked kernel sized filters - thus constituting a very deep network. Undoubtedly a formidable task, $VGG16$ took weeks to train, utilizing NVIDIA's Titan Black GPUs.

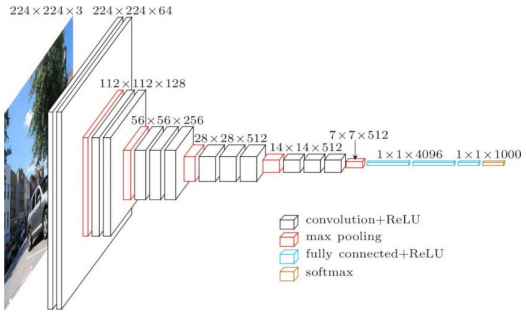

Figure 5: $VGG16$ network architecture - used to classify 1.5 million images into 1000 classes.

1.2 Model Specification

We want to utilized the trained $VGG16$ for feature extraction & thereafter train a neural net to classify the dataset by artist.

The $VGG16$ is able to classify 1000 different labels, since we only need 3 (one for each artist) we replace the last layer of the classifier with one with the desired dimensionality (mapping to the number of classes).

We only wish to train the linear (fully connected) layers.

The resulting model architecture consists of a number of pretrained convolutional layers - to effectively process the images and extract meaningful features from the data - & 3 linear fully connected layers: the first two followed by non-linear activation & dropout layers.

ReLU activations are used in order to discover non-linearities in the dataset. *Dropout* layers are used to randomly kill a portion of the nodes in training - a technique to improve generalization by preventing the model from overfitting the training set. To this aid, *dropout* is a form of stochastic regularization.

1.3 Data Pre-processing

Let's first view the data & build some intuition about how our discriminator might be able to model discrepancies between classes.

Figure 6 depicts the unique styles of each artist. One should note that the artist possess substantially different styles & as such after sufficient effective dimensionality reduction via convolution a neural net should be able to distinguish between the samples.

Figure 6: A sample from the dataset, depicting the unique styles of each artist.

To prepare the data for the model in a fashion that is both efficient & maximises the likelihood generalization we define a PyTorch data loader that sequentially processes random batches of data. The data is split into train/test/validation sets for our analysis.

When called, the test & validation sets crop the images to the appropriate $(3, 224, 224)$ dimensions required by $VGG16$ as well as converts the data to the require *torch.tensor* class. Images are also cropped slightly, beyond reshaping, to reduce superfluous bordering pixels that contain very little information.

The training set loader performs the same tasks, but additionally to increase generalization, we perform some basic data augmentation by randomly flipping some training samples - an attempt to learn spatial invariance around identical concepts/samples.

Figure 7: A sample from the dataset, after cropping & resizing the images.

1.4 Training

The model is trained entirely on the training set, the model parameters that happen to minimize the validation error are taken as the final instance. The test set is completely unused until performing the final analysis.

1.4.1 Cross Entropy Loss Function

Cross Entropy loss is used to output class probability of the network. Where there are C class, cross entropy loss for a single observation i is defined as:

$$CE_i = -log(\frac{e^{s_t}}{\sum_j^C e^{s_j}}) = -s_t + log(\sum_j^C e^{s_j})$$

Where s_t denotes the model output for observation i on the true t class. The loss is then averaged over each observation in the batch - weighted average can be used to counter class imbalance.

1.4.2 Optimizer

Stochastic gradient descent is utilized to optimize the cost function. A momentum term is added to encourage search in attempt to mitigate quickly stabilizing in a local minimum. Thus with the momentum term, the optimizer takes the form:

$$\theta^{t+1} := \theta^t - \eta \nabla Loss(\theta^t) + \mu \nabla Loss(\theta^{t-1})$$

Where θ^t are the parameter weights at iteration t, η is the learning rate, μ is the momentum coefficient & $\nabla Loss(\theta^t)$ is the gradient of the loss function taken w.r.t the parameters at time t. In our model $\eta = 0.001$, $\mu = 0.9$.

1.4.3 Regularization

Whilst we did not conduct cross validation and our loss function does not contain any regularization term (L2 norm etc) we did attempt to improve generalization by a few simple steps:

- The networks convolutional layers are pre-trained. As such the dataset in question does not have any influence on their weights & if they are able to extract meaningful features from the training set they should do so on any set.

- Dropout is the main form of explicit regularization: by randomly setting trainable weights to 0 we are able to prevent the model from simply replicating the data.

- The data is split into train/test/validation sets. The optimal model returned is the one which performs best on the validation set, not the training data. Secondary to this, the final reported results were computed by the test set - which bares no consequence in the learning process.

It is by these mechanisms that we can expect the model to generalize well to unseen data, under the assumption that the learned function approximates the true underlying target distribution.

1.5 Performance Analysis

We analyze the performance on the testing set. Since we only wish to build a discriminator that is able to distinguish between the 3 classes/artists we are primarily concerned with the accuracy of each class. Figure 8 depicts the proportions of correctly predicted samples in each class. With all classes predicted with very high accuracy, the model appears to be generalizing well to the test set. The extremely high accuracy may course concern, raising the issue of overfitting; however, in the context of the dataset we are learning to discriminate between very specific artistic styles & as such one can expect top performance on the premise that sufficient features have been extracted.

```
Test after training...

_____ Evaluating Model: _____

Evaluation completed in 0m 34s

Avg loss (testset): 0.0215
Avg acc (testset) : 0.9318

Accuracy per Class:

Alfred_Sisley : 0.94118
Albrecht_Dürer : 0.88889
Rembrandt : 0.96296
```

Figure 8: Model accuracy performance on the unseen test set after training.

Figure 9 provides an example of classifying unseen data using the model. The ground truth (known correct outcome) & model prediction is made on a random sample from the test set. In this sample, all images are correctly classified.

Figure 9: The ground truth & predictions on a random sample from the test set.

1.6 Next Steps

Our model achieved stellar results, here are some follow on ideas to continue the project:

- The *WikiArt* repository provides information on a wide range of artists. One might consider extending the application to more artists, who's style's may not be so visually independent. One might also wish to extend the input to meta-data about the art works.

- The model runs the risk of over-weighting classes as a byproduct of class imbalance. Data augmentation could be used to generate more samples from under-represented classes. Additionally the Cross Entropy Loss can be readily weighted to counteract the inconvenience of class imbalance.

- Additional regularization in the form of adapting the loss function to directly include a penalty term might yield more general results - though hyperparameter tuning may cause an unsolicited excess in computational overhead.

- Finally, one may wish to extend to generative models, attempting to learn the full data distribution over the samples, providing the capacity to generate new or manipulate current data in interesting ways.

2 PCA: Principal Components Analysis

As an alternative, simpler & theoretically robust method to solve this image classification task, one might consider performing some traditional statistical dimensionality reduction techniques & thereafter apply a learning algorithm to discriminate between the data. One simple linear variant of this is PCA.

Having strong theoretical support and simple implementation, PCA is often a good choice. One caveat, however, is that PCA may be inadequate in properly capturing the essence of the data, given it's linear restrictions.

2.1 Computational Consideration

A further problem encountered when computing PCA is the reliance on the computation of eigen value decomposition of the covariance matrix. Suppose we vectorize our images - recall our images are of dimensions $(3, 224, 224)$ - we arrive at a single image being represented by a $3 \times 224 \times 224 = 150528$ vector (each pixel representing a parameters). If one attempted to compute the covariance matrix of this quantity one would arrive at a (150528×150528) matrix, extremely large & resulting in a numerous computation & storage overheads.

Instead we employ a simple trick, & compute the eigen value decomposition in terms of the transpose of X - speeding computation and constraining memory requirements significantly. Ordinarily one would compute the eigen values on the matrix:

$$\Sigma_{p\times p} = \frac{1}{N} X^T X$$

however instead one can compute the eigen value decomposition on the substitute matrix:

$$\Sigma_{n\times n} = \frac{1}{N} X X^T$$

To this aid, we then rely on the below proof to derive the eigenvectors of the original matrix:

$$\begin{aligned} (\frac{1}{N} X X^T)\phi &= \lambda\phi \\ X^T(\frac{1}{N} X X^T)\phi &= \lambda(X^T\phi) \\ (\frac{1}{N} X^T X)(X^T\phi) &= \lambda(X^T\phi) \end{aligned} \tag{1}$$

This results in the derivation of eigen vectors on the original image space being computed as:

$$\psi = (X^T\phi)$$

Note: this trick only applies as $n << p$ however this is often the case when working with high resolution images.

2.2 Lower Dimensional Mapping

Now we perform PCA on the data & assess it's ability to capture separable features. We wish to standardize our training data. Figure **??** depicts the mean and standard deviation images for each artist. We will use these tensors for standardizing the data, however the mean images are interesting as they do appear to capture some 'theme' in the data - appearing as a low resolution generalization of the data.

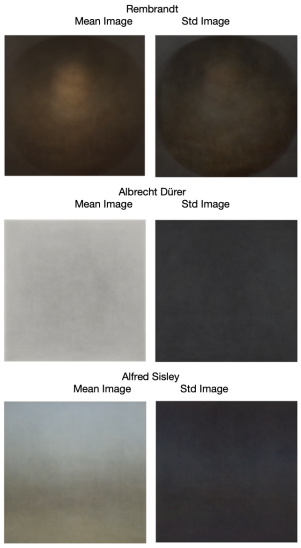

Figure 10: The mean & standard deviation images for each artist.

Conducting PCA on the data is primarily concerned with mapping to a lower dimensional manifold that maximizes the variation - information - in the reduced set. Essentially minimizing the loss:

$$\mathbf{E}[||X - f^q(X)||^2]$$

where $f(X)$ is the space of q orthogonal basis functions. We then select the first q basis vectors (those that maximise the variation captured). Figure 11 indicates that the majority of the variation in the data can be captured by very few eigenvectors - a promising sign for PCA's ability to effectively map the data into a meaningful lower dimensional manifold.

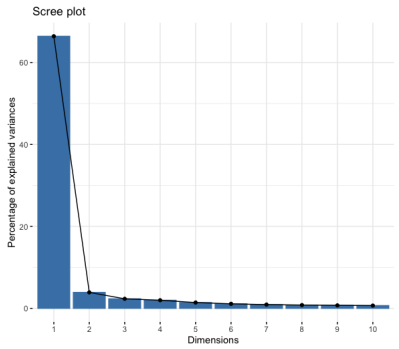

Figure 11: A scree plot of the principal components, depicting the percentage of variation explained by the orthogonal basis functions.

2.3 Visualizing the Eigenvectors

The goal is to map the data to a lower dimensional manifold that can then be fed to a discriminator algorithm to classify the data. To this aid we can visualize the separability on the first 2 or 3 PCs (principal components) to develop intuition about whether or not we can expect the data to be classified well after this transformation.

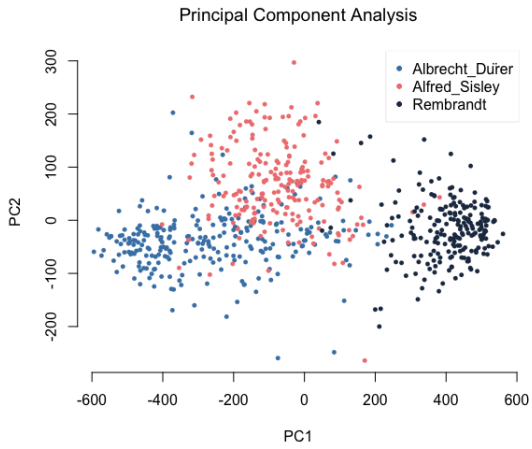

Figure 12: The data is represented by the first two principal components, coloured by the true known class.

Figure 12 depicts the separability on the first two principal components. Though some natural groupings have emerged, there is still substantial overlap. One would require a non-linear learning algorithm to effectively learn the mappings between data & labels.

One might also want to visualize the data represented on the first 3 PCs - given by figure 13 and 14 (two variations of the plot are shown as one can only develop an intuition around the 3 dimensional representation after an animated viewing). Again, it's apparent that the data would require a nonlinear mapping to separate the classes sufficiently.

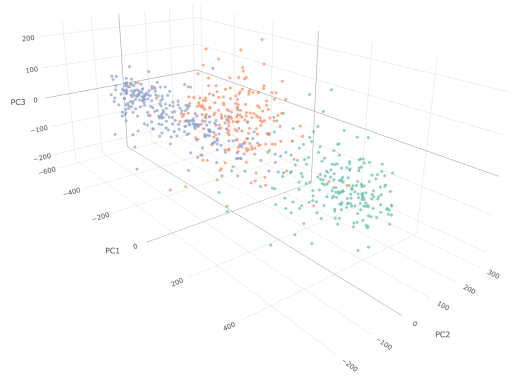

Figure 13: The data is represented by the first 3 principal components, coloured by the true known class.

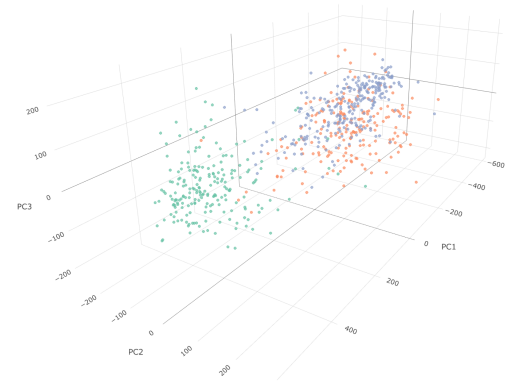

Figure 14: Identical to figure 13, viewed from a different orientation.

2.4 Reconstruct the Original Images

We now visualize the lower dimensional representations' ability to reconstruct a meaningful image. The closer the reconstruction is to the original image, the more likely it can successfully represent the original image. Essentially we're analyzing the reconstructions to develop an intuition around how to optimally trade off compression & accurate data representation.

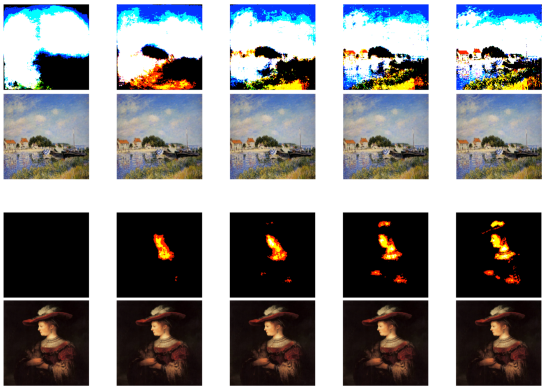

Figure 15: Two images, randomly sampled, reconstructed from the principal components. The first two rows represent the first sample image: the first row is the image reconstructed using: $\{10, 50, 100, 250, 500\}$ principal components respectively; the second row is the original image (for reference). The third & forth rows repeat this process on another random sample image.

First note in figure 15 that utilizing 250 or 500 components are almost identical (the same information captured by the 250 variant with half the storage/data complexity.

Although some images in the dataset can be reconstructed reasonably well, many samples cannot be adequately reconstructed.

2.5 Modeling with PCA

Here we consider our learnt principal components as a lower dimensional representation of the data - which we can used to effectively model the conditional distribution between classes. Whilst figure 15 allows us to build some intuition around the value of the low dimensional encoding, we can directly quantify the percentage of variation explained by the PCs & set a threshold to determine how much information is adequate to represent the input data.

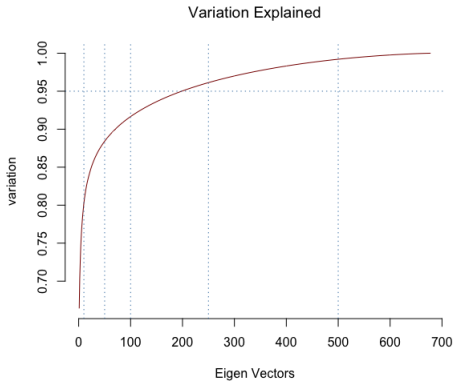

Figure 16: Variation explained by the eigen vectors - in the transposed space.

Figure 16 depicts the percentage of variation - indicating that with under 250 PCs we can represent 95% of the variation. Though larger than many other applications, we're dealing with very noisy, rich data & this represents a great reduction in dimensionality.

Now to prepare the rest of the data (training and validation sets) for modeling on the lower dimensional domain.

To prepare the training & validation sets, we scale the data & map it to the space of the PC's learnt from the eigenvectors of the training data. Note: we standardize the data using the mean and standard deviation of the training set, to avoid data snooping. Figure 17 mapping to the $q = 2 - dimensional$ PC space.

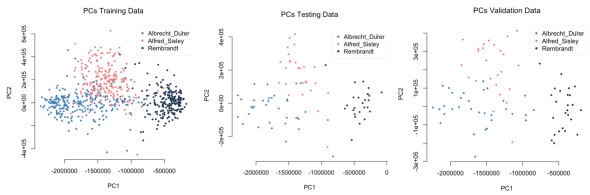

Figure 17: Mapping the training, testing and validation sets to the $q = 2$ PC space of the eigenvectors of the training set.

Figure 17 depicts the case where $q = 2$ however we can use some reasonable approximation - say $q = 50$ or $q = 100$ - as input basis functions to our machine learning model. In the next section we'll model this lower dimensional representation.

3 Multinomial Logistic Reg.

To serve as a baseline point of reference from which to contrast our nonlinear models, we fit a multinomial logistic regression to the PCs to model the log class distribution of the response variable.

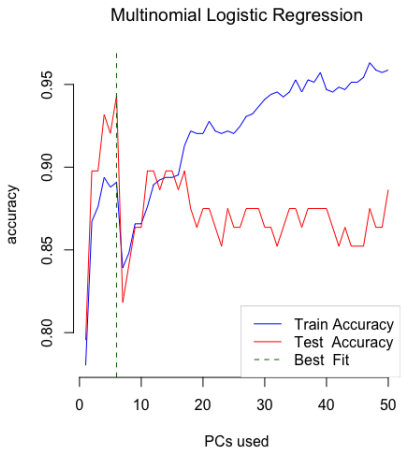

Figure 18: Training vs testing accuracy applying the multinomial logistic regression model over various numbers of PCs. Using the first 6 PC's is the optimal configuration, providing the best testing accuracy.

predicted.classes	Albrecht_Dürer	Alfred_Sisley	Rembrandt
Albrecht_Dürer	33	4	0
Alfred_Sisley	1	23	0
Rembrandt	0	0	27

Figure 19: Predictive values of each artist/class - the simple (linear) multinomial logistic regression model performs extremely well on the test data.

The multinomial logit model finds linear classification bounds between classes. The nonlinear nature of the data limits it's utility, however it's simplicity is desirable . We experimented with varying numbers of PCs as covariates, the best model - that which maximises the test accuracy, generalizing best - was found to utilize the first 6 PCs. Figure 19 shows the best model's results, achieving an overall accuracy of 94% on the test set.

It's clear that the linear hyperplane is extremely effective in separating the eigenvector decomposition of the data.

As a point of comparison, utilizing the first 2 PCs produces a test accuracy of around 88%. One should note that although the model achieved stellar results, they should be taken with caution as in this instance over the specified range (2-to-6 PCs) the test set outperforms the training set: quite possibly a consequence of favourable sampling variation/small sample size.

4 Support Vector Machines

Finally, we can use our latent PCA representation to model the data using a nonlinear approach. To this aid, we utilize a SVM (Support Vector Machine) modified for multi-class classification. Note: The package utilized ($e1071$ available in $CRAN, R$) implementation utilizes a *'one-against-one'-approach* in order to generalize the SVM (traditionally a binary classifier) to a multi-class classification problem.

It's self-evident from visualizing the PCs that the data is not (linearly) separable, as such we need to allow for some kernel trick to transform the data into a high dimensional $Z - Space$ & above that, allow for some margin of error (to account for non-separable covariates).

We wish to maximise generalization, as such we use the data splits (train/test/val) to tune the hyper parameters & select a model that we suspect will generalize best to new data.

Our $Z-space$ transformation is conducted utilizing a radial kernel, which takes the form:

$$k(x_i, x_j) = exp(-\gamma \sum_{k=1}^{p}(x_{ik} - x_{jk})^2)$$

Where γ is a hyper parameter that controls smoothness in the decision boundary - controlling variance of the model.

4.1 Optimal X-Space Dimensionality

First we want to determine the optimal number of PCs to use to maximize generalization. We fit the SVM over a range of PC's (including an additional PC on each run) & select the model that minimizes the number of support vectors needed to separate the data. Visualized in figure 20.

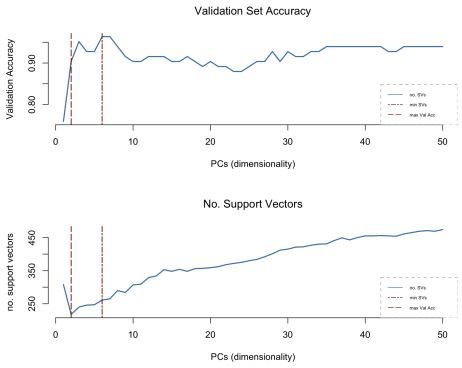

Figure 20: Selecting the optimal number of PC's (dimensionality) to use to minimize the number of support vectors (a proxy for good generalization).

4.2 Optimize Gamma

Now that we have our optimal number of PC's to minimize the number of support vectors, we can optimize our hyper parameter. Although one might be tempted to optimize for validation accuracy, we opted to again minimize the number of support vectors - following the principals of parsimony in hope of generalizing well. Figure 21 visualizes this experiment.

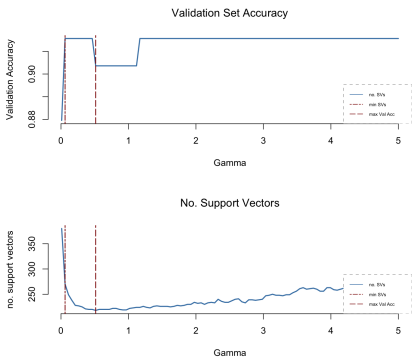

Figure 21: Selecting the optimal hyper parameter (gamma) to use to minimize the number of support vectors (a proxy for good generalization).

4.3 Results

Now that we have selected our model, we predict on the unseen testing data to assess the results. The model performs very well over all classes, yielding the predictive accuracy per class:

$$\begin{aligned} Albrecht\ Durer &= 88.24\% \\ Alfred\ Sisley &= 81.48\% \\ Rembrandt &= 96.27\% \end{aligned}$$

The model performs brilliantly over all classes. We can visualize the learned classification bounds, as seen in figure 22.

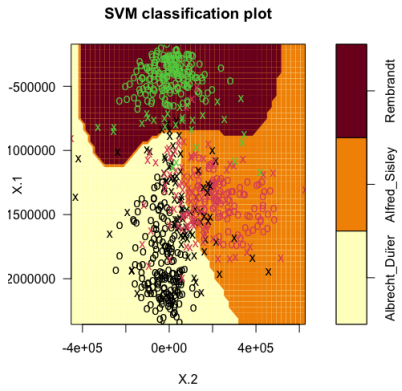

Figure 22: The learnt classification bounds over the first two PCs.

5 Conclusion

All models work well & achieve their specified objective. The transfer learning neural net is the least favourable, requiring considerable computational power & RAM to train from scratch. The two statistical models are both backed by stronger theoretical grounds & are simpler to execute.

The two statistical models are essentially equivalent, though one might expect the SVM to better generalize to new data - given the maximum bound theory - & as such is favoured.

The neural net may be optimal for more interesting extensions to the project: generative models etc.