What are you drawing? Classifying Google Quick, Draw! Doodles

Tong Li tli287@wisc.edu
Runxin Gao rgao35@wisc.edu
Kenny Jin jjin59@wisc.edu

Abstract

Image recognition, especially from scratchy and noisy data, plays a significant role in machine learning. In our project we want to classify the hand-drawn doodles, with 345 categories in total, from Google Quick, Draw!. We construct different algorithms such as the k-nearest neighbors (KNN), Logistic Regression, Random Forest,and Convolutional Neural Network (CNN), and they each obtains an accuracy of 26%, 25%, 31%, 60%. We also experimented different dimensionality reduction methods for KNN and Logistic Regression. By comparing model performances, we can gain an insight on which classifier is the most suitable for classifying doodles with a lot of categories.

1. Introduction

Quick, Draw![1] is an online game released by Google on November, 2016 where the user is prompted with a specific requirement to draw a picture in 20 seconds and the algorithm will make a prediction based on the drawing. Millions of images were collected by Google through this game, and they were utilized to make better and quicker predictions. Figure 1 showed an example of such images[2]. Such large-scale dataset leaves many imaginations to machine learning enthusiasts and encourages the rises of creative projects. For example, Quartz has explored the drawing habits like stroke order grouped by different countries and found clear associations between the drawing and their language styles [2]; others tried several different prediction models evaluated by prediction accuracy [4].

More than simply classifying doodles, such algorithms could have educational applications for language-learning toddlers as well. They can draw what they see, and learn how to spell or say it, which could accelerate the language learning process. Moreover, since doodles are somewhat sketchy, by constructing algorithms that can make accurate predictions based on sketchy images, the uses of the algorithms could be extended to convert handwritten text to computer, eliminating the need to type all the words.

In our project, rather than building separate algorithms for each category or subset of categories, we combined all categories together as our data set. Therefore, since our data set has totally 345 categories, the prediction accuracy is not very high, but is still higher than what we expected. We apply several algorithms such as KNN, Logistic Regression, Random Forest, as well as CNN, and adopt different variations of dimension reduction methods.

2. Related Work

Quick, Draw! dataset becomes popular among machine learning enthusiasts as soon as it has been released to the public. Similar to our tasks, many experts in machine learning have exploited different classification algorithms to recognize

[1] https://quickdraw.withgoogle.com/

[2] https://github.com/googlecreativelab/quickdraw-dataset

Figure 1. Example of quick draw images

Figure 2. Example drawings of teapot

doodles. Rashid applied six machine learning algorithms on the dataset in a Kaggle competition: KNN, Random Forest, Linear Support Vector Clustering (SVC), Gaussian Radial Basis Function SVC, Multi-Layer Perceptron, and CNN[5]. His goal was to classify doodles into only two categories and to analyze the performances of these models. When Rashid obtained relatively high classification acccuracies of all models, we attempt to classify doodles into 345 categories, which are all categories of the original dataset, instead of 2 categories and to check whether algorithms can consistently remain high accuracies.

Guo, WoMa, and Xu experimented three algorithms on the dataset: 1-Closest Centroid (1-CC), KNN with K-Means++, and CNN to classify doodles into all 345 unique categories[1]. They set the accuracy of 1-CC as the baseline and compared the model performances of KNN and CNN. They obtained the accuracies of 23.6%, 34.4% and 62.1% respectively; therefore, they concluded that CNN outperformed than other two algorithms. Surprised by the advantages of CNN on computer vision, we further explore two CNN models to see how CNN model can perform at the best on the image classification problem; so we tried two kinds of CNN models: LeNet-5 and ResNet-34 and tuned their hyperparameters.

Besides, there are some creative explorations on the dataset. For example, Ha and Sonnad used the public database from Quick, Draw! to find relations between the styles of drawing basic shapes and major language groups. They found that the drawing stroke orders were highly influenced by the languages and cultures. Inspired by their amazing conclusions, we try the mini-batch K-means Clustering algorithm to explore the similarities of drawing styles among all 345 distinct categories of distinct doodles.

3. Proposed Method

The data we have in hand is in the general form of typical image data, with 784 dimensions, each representing a pixel of the image. When firstly look at out data, we find the data has a bunch of uninformative features, i.e. the dataset have a lot of features with value zero for all images. Therefore, we decided to apply dimensionality reduction methods to get rid of such unuseful features and to improve computational efficiency.

We start building the classifier with KNN, which is a lazy algorithm that does not learn from the data but simply memorize. We then fit the data into penalized multinomial logistic regression with cross-validation. Then we use Support Vector Machine as well as Convolutional Neural Network as somehow more 'advanced' tools to classify the doodles. We define the accuracy score by the proportion of the number of doodles correctly classified.

$$ACC = \frac{\text{Number of Correctly Classified}}{\text{Total number of Doodles}}$$

We randomly split our data into 80% training and 20% testing.

3.1. k-Nearest Neighbors

The first method that comes in mind when doing classification tasks is the k-Nearest Neighbors algorithm. It is a lazy algorithm that simply memorize the data. KNN algorithms takes two parameters: k, the number of neighbors to be considered, and p, the parameter for the Minkowski distance function. Minkowski distance is defined as

$$D(\mathbf{X}, \mathbf{Y}) = (\sum_{i=1}^{n}(|x_i - y_i|)^p)^{\frac{1}{p}}$$

And this distance corresponds to the Manhattan Distance when p = 1, and Euclidean Distance when p = 2.

When classifying a data point, KNN calculates the k nearest neighbors by the Minkowski distance defined by p, get their labels, and then determine the label of the new data through majority (or plurality) voting. To obtain the optimal k, we conduct 5-fold cross validation on the training set. To wit, the training set is split into 5 portions, and the algorithm is fit five times. Each time the KNN use 4 portions as training and the remaining one as validation. The final accuracy is obtained by averaging the performance of the 5 validation sets in each fitting. In addition, since KNN can be susceptible to the curse of dimensionality, we apply two dimensionality reduction methods: Principal Component Analysis and Autoeocoder. The number of optimal components is also obtained through cross-validation. The number of hidden layers in Autoencoder is the same as PCA's number of components.

3.2. Multinomial Logistic Regression

We then fit the data into multinomial logistic regression with L1 penalty (LASSO). Multinomial logistic regression returns the probability of the label equals to k with the following equation

$$\mathbf{Pr}(Y^{[i]} = k) = \frac{e^{\beta_k \mathbf{X}^{[i]}}}{\sum_{i=1}^{K} e^{\beta_i \mathbf{X}^{[i]}}}$$

To prevent this model from overfitting the data, we apply L1 penalty to the model, with coefficient λ obtained through 5-fold cross validation. We also use the two dimensionality reduction methods as mentioned above to accelerate computational time.

3.3. Random Forest

Random forest is an ensemble learning method that combines multiple decision trees to make the final prediction. Comparing to using a single decision tree, the method of random forest is usually more robust to overfitting, thus this model will perform better than decision tree on the test set. Therefore, we decided that this model will also be included in our experiment.

Random Forest models can be useful for feature selection, thus when using random forest for prediction, it might be unnecessary to adopt dimensionality reduction techniques before feeding the data into the model. However, we still wanted to compare the prediction performance of random forest between differently manipulated datasets. Thus, we will still use the 2 dimensionality reduction techniques (PCA, Autoencoder) and compare the prediction result with that of the original dataset.

3.4. Convolutional Neural Network

Different than all the traditional machine learning models described above, the Convolutional Neural Network (CNN) is a class of deep neural networks that are especially powerful in the field of image recognition. There are various structures of CNNs, but they will always have convolutional layers and pooling layers in addtion to fully connected layers. The special structure of CNNs might be able to capture hidden features of images, thus making them perform better than many other machine learning models when doing image classification.

There are many different types of CNNs, and we selected specifically 2 of them. One is called LeNet-5[6], which is a classical CNN model that has a relatively simple structure, and another is called ResNet-34[3], which is a newer model

whose structure is much more complicated. Commonly regarded as the pioneer of CNNs[3], LeNet-5 contains 7 layers, 3 of which are convolutional layers. ResNet-34, however, is a much larger neural net that contains 34 parameter layers[4], and its architecture is very different than LeNet-5. Both models can be used for image classification, thus we selected these 2 models for our project and wanted to compare their performance with other traditional models.

4. Experiments

4.1. Dataset

The original data set we have consists of 345 Numpy Bitmap files from Google, each .npy file consists of thousands of gray scale images for each category. Google has already pre-processed the data in .npy files by only keeping image data and centering the image with 28 * 28 pixels. Because it is not practical to run the whole data set on our computer, we decide to take a random sample of 1000 images from each category, and combined them to one single .npy file, which has 345,000 images in total. In fitting the models, we split our data set as 80% for training and 20% for testing using $train\_test\_split$ in scikit-learn stratified on labels. Each image is represented as having 784 dimensions, with values representing the intensity of each pixel, ranging from 0 - 255. We apply max-min normalization to the image data, and the range thus becomes 0 - 1. In addition, the label is represented as 0 - 344 for each category, rather than the exact text label. Global random seed is set at 123.

4.2. Dimensionality Reduction

PCA Since image is centered, we find that there are a lot of features that are all zeroes for all images, thus making them uninformative in model prediction. Therefore, we decide to use Principal Component Analysis to get rid of uninformative features and reduce dimensions. We firstly graphed the explained variance of the feature we find that 256 features already explain 100% of the variance in the data, and that there does not seem to be a much difference between 100, 200, and 256. Thus we apply cross-validation to find the optimal number of components for KNN and `n_component` = 100 is suggested. We fit the PCA on training set, and transform training and test sets to be of 100 dimensions.

Autoencoder We also applied another method of dimensionality reduction called autoencoder. Since PCA is only an linear method, we are concerned that it may not preserve the information in the images very well. On the contrary, the method of autoencoder might perform better than PCA regarding the task of dimensionality reduction. The autoencoder we used for this project is a 3-layer neural network, with one input layer, one hidden layer and one output layer. The input layer takes in the original image vector with 784 dimensions, and the output layer reconstructs the image back to 784 dimensions. We set the number of nodes in the hidden layer to be 100 to make it consistent with PCA. We set the learning rate as 0.05, the random seed as 123 and the batch size as 1000. After 15 epochs of training, the autoencoder can already reconstruct the images pretty well. Then we rerun the autoencoder on all the examples (1000 images for each category) and extracted the hidden layer vectors (100 dimensions) as a new dataset whose dimension was reduced.

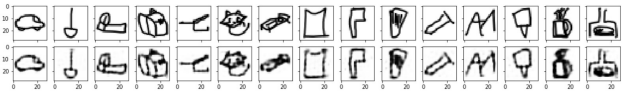

Figure 3. Original Image VS Image reconstructed from Autoencoder

[3]https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/cnn/cnn-lenet5-quickdraw.ipynb

[4]https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/cnn/cnn-resnet34-quickdraw.ipynb

4.3. k-Nearest Neighbors

To obtain the optimal choice of k and `n_components`, we use the GridSearchCV function with 5-fold cross validation. We make a pipeline to determine the combination. For PCA the number of components to be chosen from are [100, 200, 256], and for k the range is [5,100] with step = 5. Since our data set is too large and GridSearchCV failed to return the result even after one and a half days, we decided to run it on a smaller subset with 200 images for each category. This greatly enhanced the computational time. We then use Autoencoder to compress the data into smaller dimensions and fit the kNN model with optimal k obtained previously, and compare the results. We abort training kNN with full dimensions after the computation takes more than one day.

4.4. Multinomial Logistic Regression

Because we have so many features in our data, we use L1 penalization for the logistic regression to avoid overfitting of the data. We adopt the 5-fold cross validation to determine the optimal strength of penalty (λ).

We fit the model with three different training sets: one with dimensions reduced by PCA, one with dimensions reduced by Autoencoder, and one with full 784 dimensions to compare model performance. As suggested in Sci-Kit Learn's documentation, we choose SAGA as our solver. We also change the tolerance level to 0.05 because the default tolerance makes it extremely slow for the model to converge.

4.5. Random Forest

To make a selection regarding the best parameters for random forest, GridSearchCV function could be used. However, When we tried to use this method for hyperparameter selection, the memory error keeps emerging. Thus, we have to use 3-way holdout method to try out different hyperparameters and check the performance of the model on the validation set. After this procedure, we decided to use 500 estimators and sex the max_depth of each decision tree to be 16. Then we fit the random forest model with the hyperparameters above using the original dataset, PCA-transformed dataset and autoencoder-transformed dataset, respectively. When fitting the final models, we merge the training and validation set back to a complete training set and use it for model training.

4.6. Convolutional Neural Network

We selected LeNet-5 and ResNet-34 for the prediction task. The implementation of these 2 models is adapted from Professor Raschka's code[5] for deep learning models. For both LeNet-5 and ResNet-34, we set the batch size equal to 128. The number of epochs for training are set differently for these 2 CNNs. We set NUM_EPOCHS = 50 for LeNet-5, but set it to 10 for ResNet-34, since the structure of ResNet-34 is more complicated and it needs more training time for each epoch. We also set the RANDOM_SEED = 123 for the training of each CNN.

For deep neural networks, the learning rate settings are important. The cost function of a CNN could fail to converge under an improper setting of learning rate. To make sure each model converges, we tried different learning rates for each model. For LeNet-5, we set the learning rates to be 0.0001, 0.0005 and 0.001. For ResNet-34, we set the learning rates to be 0.001, 0.005. By checking the plots of the cost vs number of epochs, we are able to see if the training of CNN converges.

5. Results and Discussion

For KNN, the optimal combination is `n_components` = 100 and k = 15. Therefore when using autoencoder, we also compress the data into 100 dimensions. For logistic regression, three models actually have different penalty strength. The optimal penalty strength

[5] https://github.com/rasbt/deeplearning-models

Model	Accuracy
KNN_PCA100	33.19 %
KNN_Autoencoder	26.13 %
Logistic_PCA100	24.67 %
Logistic_Autoencoder	23.72 %
Logistic_784	23.96 %
Random Forest_PCA100	30.87 %
Random Forest_Autoencoder	28.33 %
Random Forest_784	31.25 %
CNN_LeNet5	46.58 %
CNN_ResNet34	59.87 %

Table 1. Test Set Accuracy for Different Models

returned by LogisticRegressionCV is 100, 10000, and 1 for PCA-100, Autoencoder-100 and full dimensions, respectively. Table 5 shows the test set accuracy we obtain from each model.

5.1. k-Nearest Neighbors

By adopting GrivSearchCV on a subset of the training data, the optimal number of neighbors to be considered is 15, and the best number of components is 100. By reducing the data into 100 dimensions using PCA and Autoencoder, prediction accuracy on test set is 33.19 % and 26.13% respectively. Because we have too many data with large dimensions, KNN takes several hours to be trained. The discrepancy of the test set accuracy is quite large for different dimensionality reduction methods.

5.2. Multinomial Logistic Regression

By training the logistic regression with their optimal L1 penalty strength obtained from cross-validation, the test accuracy is 24.62%, 23.72%, 23.96% for PCA-100, Autoencoder-100, and full dimensions. The test set accuracy is not quite different for these three models, but it takes much less time to fit the regression with reduced dimensions.

5.3. Random Forest

After training on the original dataset whose dimensionality is not reduced, the random forest model gives a test set prediction accuracy of 31.25%. Using PCA-transformed dataset for training, the prediction accuracy becomes 30.87%, which is slightly lower than that of the original dataset. Training with the autoencoder-transformed dataset produces the lowest testing set accuracy among all 3 results, which is 28.33%. It seems that it is unnecessary to perform dimensionality reduction before feeding the data into this random forest model.

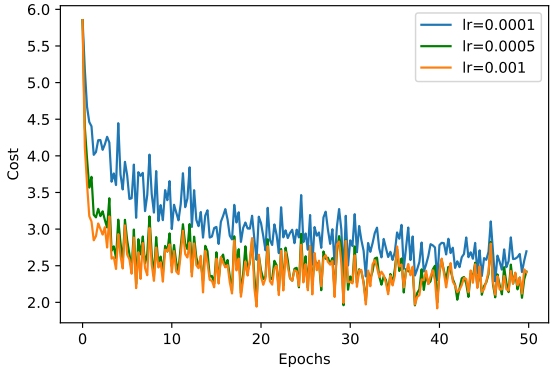

Figure 4. LeNet-5 cost function plot for different learning rates

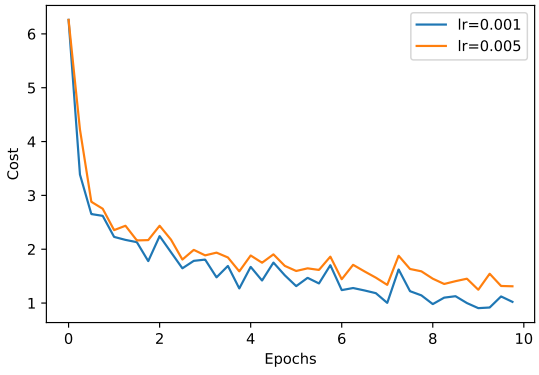

Figure 5. ResNet-34 cost function plot for different learning rates

5.4. Convolutional Neural Network

After the training of each neural network, we observed that for LeNet-5, the model will converge under all of the 3 learning rates (0.0001, 0.0005 and 0.001). However, when learning rate is 0.0005, the test set prediction accuracy is 46.58%, the highest of all these 3 learning rates. For ResNet-34, the model seems to be converging under both learning rates (0.001, 0.005), although the learning rate 0.001 will give a better test set prediction accuracy of 59.87%. The training of ResNet-34 (10 epochs) clearly takes more time than that of LeNet-5 (50 epochs).

5.5. Mini batch K-Means Clustering

We choose K-means Clustering group pictures of their characteristics considering expensive computational cost of K-means Clustering. Then we use the Elbow method select the optimal hyperparameter K. From Figure 6, 50 clusters are a reasonable choice since the distortion is dramatically decreasing after 50 clusters.

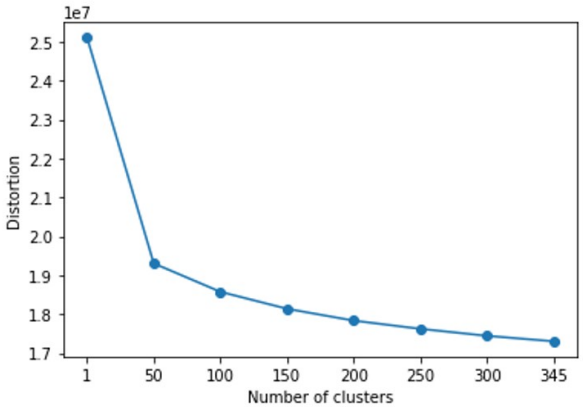

Figure 6. Distortions with each number of clusters

5.6. Discussion

As expected, CNN returns the best test set accuracy for doodle classification, while logistic regression does not perform so well. From this experiment, we can see that deep learning models are indeed powerful for the task of image recognition. For other models, since features of images cannot be properly processed, the prediction performance is much lower. By looking at the discrepancy between training accuracy and test set accuracy, we see that for some model overfitting occurs.

For the exploratory data analysis part,we listed the top 10 categories of drawings with most similar drawing styles are ”line,” ”headphones,” ”rainbow,” ”pear,” ”triangle,” ”t-shirt,” ”The Eiffel Tower,” ”rifle,” ”tennis racquet,” and ”wine glass.” These categories basically are common in daily life and usually have simple shapes so that people tend to agree with each other on how to draw them using a few strokes. People tend to simplify abstract concepts by drawing. For example, people use a small number of strokes to describe the relatively complex concept: ”animal migration,” which is similar to the style of ”rain”. We also find some unexpected categories share drawing styles like the categories ”skull” and ”mushroom.” (see Figure 7)

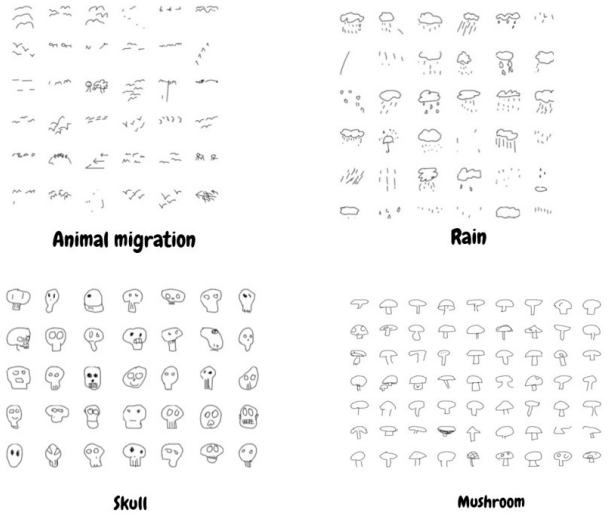

Figure 7. Interesting findings

There are some limitations of this study. Due to the time constraint, we are not able to include more data for this project. If more data were used, the results could be more reliable. Besides, we did

not perform enough hyperparameter tunings for each model, thus the performance of each model might not be optimal. In the original dataset, additional time-stamp data are provided. This dataset contains people's order of drawing each stroke, thus might be useful for improving prediction performance. However, we did not include this part in our project due to the time constraints.

6. Conclusions

We can conclude that for the task of doodle classification, the CNN models have superior performance, and the ResNet-34 is the best model regarding the prediction accuracy.

For future experiments, we can include more data and perform more hyperparameter tuning in order to improve the prediction performance of each model. We can further include the time-stamp data in the project and try some interesting models, such as recurrent neural networks (RNN), to predict using this part of data.

7. Acknowledgements

We would like to express our thanks to Prof. Raschka for kindly providing us with suggestions about this project and code examples for the implementations of CNNs.

8. Contributions

Each group member contributed evenly to data cleaning, modeling, and writing of this final report.

References

[1] Guo, K., WoMa, J., Xu, & E. Quick. Quick, Draw!Doodle Recognition, 2018. [Online; accessed 18-December-2019].

[2] Ha, Thu-Huong, and Sonnad, Nikhil. How do you draw a circle? We analyzed 100,000 drawings to show how culture shapes our instincts, 2017. [Online; accessed 20-October-2019].

[3] He, K., Zhang, X., Ren, S., & Sun, J. Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Irie, Keisuke. quick,draw! prediction model, 2017. [Online; accessed 20-October-2019].

[5] Rashid,H . Let's play with Quick Draw!!, 2018. [Online; accessed 18-December-2019].

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, November 1998.