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

Tong Li tli287@wisc.edu
Runxin Gao rgao35r@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, and Convolutional Neural Network (CNN), and they each obtains an accuracy of . 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 11 showed an example of such images[2]. Such large-scale data set leaves many to the imagination of 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 [**?**]; others tried several different prediction models evaluated by prediction accuracy [**?**].

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 345 categories in total, the prediction accuracy is not very high, but is still higher than what we expected. We apply several algorithms such as KNN, Logistic Regression and CNN, and adopt different variations of dimension reduction methods.

Figure 1. Example of quick draw images

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

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

Figure 2. Example drawings of teapot

2. Related Work

Quick, Draw! dataset is popular among machine learning enthusiasts once it has been released. Similar to our tasks, many experts of machine learning compared different classification algorithms to recognize doodles. Harun-Ur-Rashid experimented on the dataset in the Kaggle competition [1]. He applied six machine learning algorithms: KNN, Random Forest, Linear Support Vector Clustering (SVC), Gaussian Radial Basis Function SVC, Multi-Layer Perceptron, and CNN. His goal was to classify doodles into only two categories and to analyze the performances of these models. When Harun-Ur-Rashid obtained relatively high classification accuracies of all models, we attempt to classify doogles into 345 categories which are all categories of the original dataset and to check whether algorithms can remain consistently 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 [2]. 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 for the computer vision prombles; so we tried two kinds of CNN models: LeNet-5 and ResNet-34 and tuned their parameters.

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 to use another machine learning algorithm mini-batch K-means Clustering 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

3.4. Convolutional Neural Network

Describe the method(s) you are proposing, developing, or using. I.e., details of the algorithms may be included here.

4. Experiments

4.1. Dataset

The original data set we have consists of 345 Numpy Bitmap files from Google, each .npy consists of thousands of gray scale images for each category. Google has already preprocessed 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 Sci-Kit 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.

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.

Model	Accuracy
KNN_PCA100	33.19 %
KNN_Autoencoder	26.13 %
Logistic_PCA100	24.67 %
Logistic_Autoencoder	23.72 %
Logistic_784	23.96 %
Random Forest	fill %
CNN_LeNet5	47.25 %
CNN_ResNet34	59.96 %

Table 1. Test Set Accuracy for Different Models

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

4.6. Convolutional Neural Network

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 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

5.4. Convolutional Neural Network

5.5. Discussion

As expected, CNN returns the best test set accuracy for doodle classification, while logistic regression does not perform so well. By looking at the discrepancy between training accuracy and test set accuracy, we see that for each model overfitting occurs.

6. Explore drawing styles using mini batch K-Means Clustering

6.1. Methodology

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 3, we choose 50 clusters since the distortion is dramatically decreasing after 50 clusters.

Then we

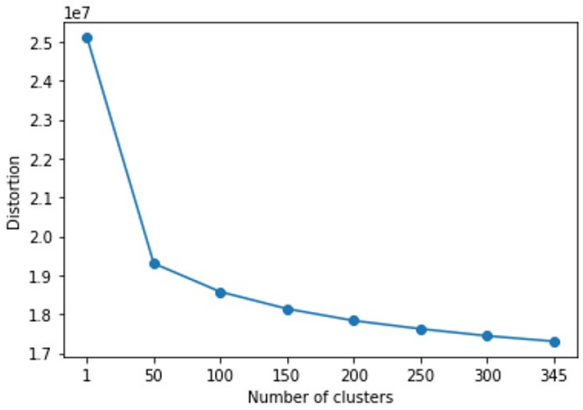

Figure 3. Distortions with each number of clusters

6.2. Discussion

7. Conclusions

Describe your conclusions here. If there are any future directions, you can describe them here, or you can create a new section for future directions.

8. 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.

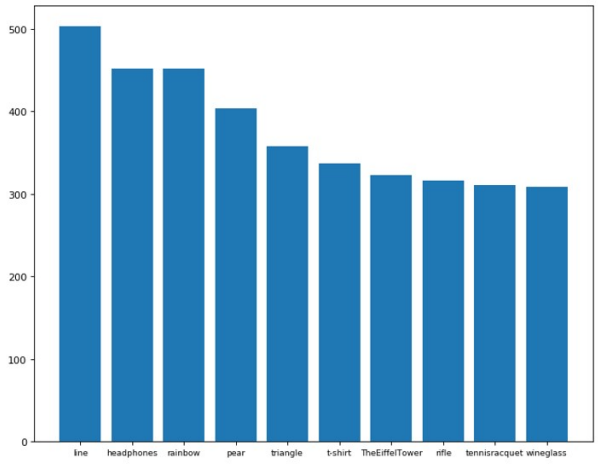

Figure 4. Top 10 categories among all clusters

Figure 5. Examples of

Figure 6. Examples of

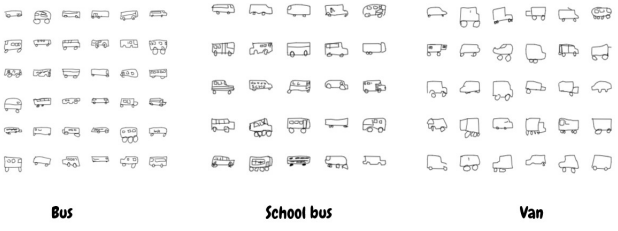

Figure 7. Examples of

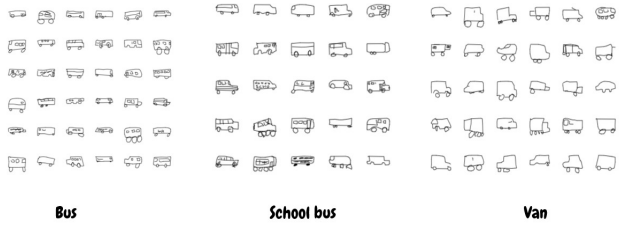

Figure 8. Examples of

Figure 9. Examples of

Figure 10. Examples of

Figure 11. Examples of

9. Contributions

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

10. References

[1]: Rashid, H. Retrived from https://www.kaggle.com/harunshimanto/lets-play-with-quick-draw.

[2]Guo, K., WoMa, J., Xu, E. Quick, Draw! Doodle Recognition. Retrieved from http://cs229.stanford.edu/proj2018/report/98.pdf.

[3]: Sonnad, N. (2017, June 15). How do you draw a circle? We analyzed 100,000 circles to show how culture shapes our instincts. Retrieved from https://qz.com/994486/the-way-you-draw-circles-says-a-lot-about-you/.