Automatic Recognition of Pictured Dishes in Food-101

Abishek Kumar
Department of Computer Sciences
University of Wisconsin-Madison
akumar225@wisc.edu

Abstract

Classification in machine learning is a supervised learning approach in which a computer program learns from the data given to it and makes new predictions. This input data called the training data contains characteristics or features about the samples under observation along with a label for each of them. The core goal of classification is to learn a mapping between the input features and labels and use that mapping to predict the labels of new unseen data. In this project, we will address the problem of automatically recognizing pictured dishes on Food-101 dataset using three different approaches and evaluate their performance. The dataset consists of 101 food categories with 101,000 images, 1000 images for each category split into 750 training images, and 250 test images. The training data is left intentionally unclean and can contain noise in both features and labels. We will study the performance of Linear Regression, KNN, and Neural Networks on this dataset and propose a recommended approach based on our results.

1 Introduction

Classification is an important tool in today's world where big data is used to make all kinds of decisions in economics, medicine, and more. For example, detecting spam in emails can be identified as a classification problem with two classes as spam and not spam. The problem of object detection and classification is a widely researched topic in machine learning due to its broad range of applications. There is an array of different algorithms that have been developed over the years to solve this problem. Food classification in particular has many applications like helping patients track their calorie intake, auto-organizing of pictures in mobile phones, etc. But, unlike other image recognition tasks wherein we have definitive features separating each class, the problem of food classification is unique due to its high intra-class variability. Lighting, orientation, and the very realization of a recipe are some of the factors which can contribute to this variability.

In this project, we will try to understand the performance of simple classification algorithms like Linear Regression, KNN for the food classification task and compare their performance against more sophisticated algorithms like Neural Networks. We will use the Food-101 dataset for training our classifiers and evaluate their performance using the holdout method. Finally, we will propose a recommended algorithm based on our results.

2 Dataset: Food - 101

The dataset consists of 101 food categories with 101,000 images, 1000 images for each category split into 750 training images and 250 test images. Both the training and test images have been downloaded from foodspotting.com. The dataset has the top 101 food categories from the website. The test images have been manually cleaned, while the training images were left intentionally unclean and have some amount of noise. This noise comes mostly in the form of intense colors and sometimes

mislabelled samples. All images have been re-scaled to have a maximum side length of 512 pixels and smaller images have been discarded.

3 Algorithms

3.1 Linear Regression

In statistics, linear regression is a method of modelling relationship between a scalar response (dependent variable) and one or more explanatory variables (independent variables). This relationship is modelled as a linear predictor function whose unknown parameters theta are estimated from the data. There are many flavors of Linear Regression like Least Squares Regression, Ridge Regression, Lasso Regression, etc. In this project, we will be using Linear Regression with Tikhonov regularization, also known as Ridge Regression for our classification. The motivation behind choosing Ridge Regression is due to its robustness to noise in the input dataset. Since our training data has noise in both features and labels, this method would give better performance compared to ordinary Least Squares approach.

3.2 K Nearest Neighbours

KNN is a type of lazy learning algorithm where labels are assigned to a new sample based on the labels of it's K closest neighbors. It's a simple algorithm where the training step involves just memorizing or storing the input features in an n-dimensional space. Classification involves plotting the new data point in the same n-dimensional space and choosing the class based on voting. The class which gets the most votes in the K Nearest Neighbors of the new sample will be assigned to the new sample.

3.3 Neural Networks

A neural network is a series of algorithms that endeavours to recognize the underlying relationships between data through a process that mimics the human brain. In recent times, neural networks have surpassed all previous benchmarks in the tasks of object recognition and classification. This is mainly due to the ability of neural networks to learn features of their own without any feature engineering. In this project, we will experiment with a neural network of 3 - 5 layers and also apply transfer learning and understand how it performs compared to the other two algorithms.

4 Implementation

4.1 Preprocessing

As a pre-processing step, we first separated the training and testing images into two different folders. Then, we resized each image in the dataset to the size of 128 x 128 x 3. We chose this size mainly due to memory constraints. Initially, we tried to experiment with the size of 299 x 299 x 3. But the number of pixels in this case was huge - 268,203. This meant that the model had to learn 268,203 weights for fitting the data. This led to memory issues even on a machine with 500GB of RAM. Hence, we decided to resize the images to 128 x 128 x 3 which gave a fair trade-off between image quality and feasibility of computation. We used python's cv2 library for resizing the images.

4.2 Experimental Setup

The experiments were done on ml.p3.16xlarge machine on Amazon AWS Sagemaker. The machine has the following configuration - 64 cores, 500GB of RAM and 250GB of hard disk space. We started with the following machines at first - 16GB RAM, 4 cores and 31GB RAM, 8 cores. But, both these machines were not able to handle the huge size of the dataset and hence we decided to go for a bigger machine. The entire implementation was done in Python and we used Python's scikit-learn library for training and evaluation of the models.

Table 1: Training and Testing Accuracy for each classifier using Ridge Regression

Sno	Loss Function	Lambda	Training Accuracy	Testing Accuracy
1	Squared-Loss	0.001	1.08%	1.36%
2	Log Loss	0.001	4.8%	1.4%
3	Hinge	0.001	3.8%	1.5%
4	Hinge	10	4.1%	1.9%
5	Hinge	30	3.2%	1.5%
6	Hinge	10	**4.8**%	**3.3** %

4.3 Approach

4.3.1 Linear Regression

First, we performed the classification using ridge regression. Sklearn library provides a direct implementation for this - RidgeClassifier. This implementation loads the entire dataset into memory and trains the model. We could not use it directly for this dataset as it was running into memory issues. Hence, we decided to train the model iteratively using Sklearn's Stochastic Gradient Descent Classifier (SGDClassifier). We split the dataset into batches of size 750 and iteratively trained the model. For testing purpose, we split the data into batches of size 250 and computed the accuracy in each batch. Finally, we averaged the accuracy across all batches to produce overall testing accuracy.

4.3.2 K Nearest Neighbours

Next, we decided to use the KNN Algorithm for classification. For this, we used KNeighborsClassifier from scikit-learn library. Since the training was taking a lot of time (>17 hrs) on the entire dataset, we decided to train and test the classifier only on a subset of data. We randomly sampled 7500 images from the training data and used it for training. We ensured that the training data had samples from all classes and tested it using a set of 2500 randomly sampled test images.

4.3.3 Neural Networks

Finally, we used Neural Networks for the classification task. We decided to use Tensorflow's keras library for implementing the neural networks due to its simplicity and ease of use. We used the Model class from the keras library for constructing the different neural network architectures and used the fit and the evaluate methods for training and testing respectively. For training the model, we used the entire training set of 75,750 images. And, we evaluated the performance on the entire test set of 25,250 images.

5 Results

5.1 Linear Regression

We trained 5 different classifiers with different settings for the loss function and the regularization parameter lambda. These classifiers were trained on a subset of data - 7500 training images and 2500 testing images to quickly identify the optimal setting for the loss function and the regularization parameter. The training and testing images were randomly sampled from the original dataset. Once we identified the optimal setting, we also trained a 6th classifier on the entire dataset using the optimal setting. Table 1 gives the training and testing accuracy for each classifier.

5.2 K Nearest Neighbours

We trained 7 different classifiers with different settings for K values. Figure 1 shows the plot of K values vs accuracy on the test set. From the figure, we can see that the best performance is achieved with K value as 38. But, we can clearly see that the accuracy of all the KNN classifiers were very poor (<3%).

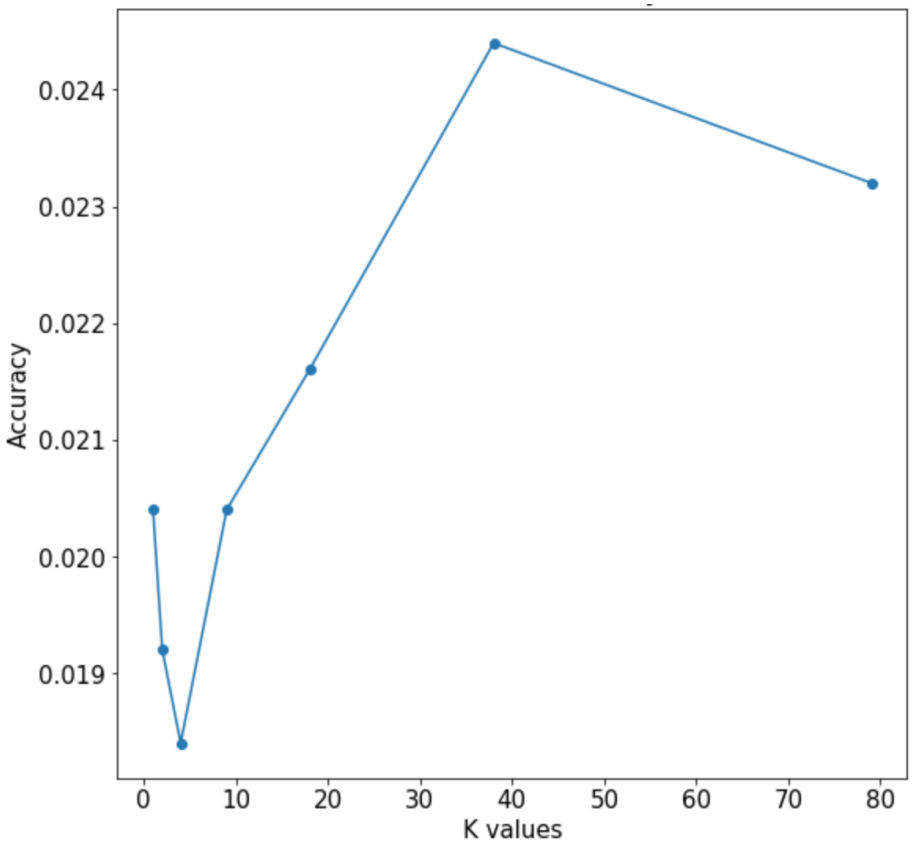

Figure 1: K-values Vs Accuracy

Table 2: Training and Testing Accuracy for the different Neural Network models

Sno	Model	Total Parameters	Training Accuracy	Testing Accuracy
1	FC Model (4 Layers)	14,806,401	8.60%	7.70%
2	FC Model (10 Layers)	1,644,415,338	10.10%	5.70%
3	CNN Model (128 x 128 Input)	9,602,917	98.84%	15.28%
4	CNN Model (300 x 300 Input)	46,073,701	99.90%	13.90%
5	VGG16	57,286,565	90.46%	43.75%
6	InceptionV3	156,125,061	90.91%	**76.54**%
7	ResNet50	233,407,461	99.98%	54.48%

5.3 Neural Networks

We trained 7 different neural network models for the classification task. We started with a simple architecture consisting of only fully connected layers for the first two models. Since the performance was poor using only the fully connected layers, we then decided to try out a model with convolutional layers and max-pooling layers. We trained models 3 and 4 using the same CNN architecture but with different input sizes of (128, 128) and (300, 300) respectively. Since the performance improved after the addition of the convolutional layers, we decided to experiment with transfer learning to improve the performance even further. For the last 3 models, we did transfer learning using the VGG16, the InceptionV3, and the ResNet50 models respectively. The detailed architecture of each model can be found in the Neural Network section of the jupyter notebook in Section 7. For the first 3 models, we scaled both the training and testing images to the size of (128, 128) and for the last 4 models, we used the input size of (300, 300). We trained all the models using the *sgd* optimizer and the *sparse_categorical_crossentropy* loss function. Table 2 gives the accuracy scores for each of the neural network models.

6 Conclusion

From the results section, we can see that the Neural Network models outperform both the Linear Regression and the K Nearest Neighbours models by a significant margin. This is due to the fact that the first two models consider each pixel as an individual feature and ignore the spatial correlation between the pixels. Also, in the case of linear regression model, the number of training samples for each class (750) is very less compared to the number of features (49,152). This is an under-determined system and hence the model was not able to learn the weights accurately. Within the neural network models, we can see that the CNN models outperform the fully connected models and the pre-trained models outperform the CNN models. This is because the CNN models consider the spatial correlation between the pixels and the pre-trained models have already learned the weights from other image classification datasets. Hence, they are able to generalize well to this dataset as well. Our best performing model gave an accuracy score of 76.54% whereas the state-of-the-art accuracy score for the Food-101 dataset is 96.18%. We believe this discrepancy can be further reduced by using image augmentation to increase the training data size and by tuning the hyper-parameters so that the neural network is able to generalize well than it is able to do today.

7 Final Deliverables

The implementation of each algorithm along with the final report is available in this Github Repository - Automatic-Recognition-of-Pictured-Dishes-in-Food-101.

8 References

[1] Bossard, L., Guillaumin, M., & Van Gool, L. (2014) Food-101 – Mining Discriminative Components with Random Forests, *European Conference on Computer Vision*.

[2] S. Xiang, F. Nie, G. Meng, C. Pan & C. Zhang, "Discriminative Least Squares Regression for Multiclass Classification and Feature Selection," in IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 11, pp. 1738-1754, Nov. 2012, doi: 10.1109/TNNLS.2012.2212721.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[4] Simonyan, K. and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." CoRR abs/1409.1556 (2015): n. pag.

[5] Szegedy, Christian et al. "Rethinking the Inception Architecture for Computer Vision." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 2818-2826.