**Statistical Learning**

**Farhad Salemi**

11/05/2019

**Introduction**

We were tasked to formulate a binary classification problem using CIFAR-10 dataset. Due to the similarities in the image background between Frogs and Ships we decided to define our question as follows:

**Which model can best predict whether an image is a ship or a frog?**

Our initial thought was that the background similarities can fool a simple/unoptimized model, and it will not able to distinguish between the two images accurately.

1. **Get the Data**

CIFAR-10 image dataset is a widely used dataset for machine learning and image recognition. The data consists of 10 classes of images: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each of the images are 32x32 color pixels with 6,000 images per class, and this totals to 60,000 images.

The original dataset included 50,000 images for training and 10,000 for testing. Since in our question we are only interested in the images of Frogs and Ships, we selected these two classes of images from the original test and train dataset and discarded the other eight classes. Therefore, we ended up with 10,000 total images in train set (5,000 images of Frogs and 5,000 images of Ships). Similarly, we got 2,000 total images in the test dataset (1,000 images of each) for model evaluation.

We then decided to divide this training dataset into training and validation so that we can validate our model during the model training/selection phase. The division was done with .5 ratio and eventually we got:

print('X shapes: ', X\_train.shape, X\_valid.shape, X\_test.shape)

print('y shapes: ', y\_train.shape, y\_valid.shape, y\_test.shape)

X shapes: (5000, 32, 32, 3) (5000, 32, 32, 3) (2000, 32, 32, 3)

y shapes: (5000, 1) (5000, 1) (2000, 1)

Also, in this step we changed the label in the output parameter (y) from original Frog/Ship to 0/1 so that we can work with binary data.

1. **Explore the Data**

As it can be seen in the above printout, the data in its original form is a four-dimensional dataset. The first dimension is the number of the image. The second and third are the length and width of a 32X32 image and the fourth dimension is for color (red, green, and blue). So, for every image we have a total of 3072 entries which is stored in a three-dimensional numpy array of 32 X 32 X 3. This shape is suitable for Convolutional Neural Net, but since we are starting with some other algorithms in this project, we have to reshape this format first which will be discussed in the next section.

There was no NA value in this dataset.

1. **Preprocessing**

As it was mentioned, the first step in preprocessing was to reshape the data, so that it could be used in non-CNN algorithms. The final reshaped data ended up in the following format:

print('X shapes: ', X\_train.shape, X\_valid.shape, X\_test.shape)

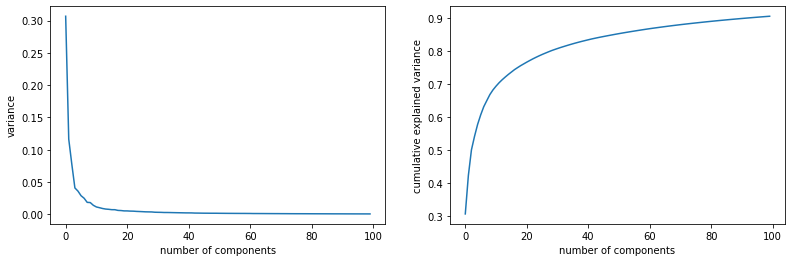
print('y shapes: ', y\_train.shape, y\_valid.shape, y\_test.shape)

X shapes: (5000, 3072) (5000, 3072) (2000, 3072)

y shapes: (5000,) (5000, 1) (2000, 1)

Next, we had to standardize the data and this is because we would fit a KNN model which needs standardization.

Finally, due to the fact that we have too many features for every image with a lot of covariances between them which can easily overwhelm the models and give us non-optimum results, we decided to use Principal Components Analysis (PCA) and decompose the features into a smaller set. We performed this task by transforming the original feature space to a lower dimensional space that still captures most of the information. Drawing the Scree plot of the principal components showed that by using 100 components, we will have %90 of variation explained:



So, we used 100 components for the rest of this project (except for CNN). This process reduced the size of our data to:

X shapes: (5000, 100) (5000, 100) (2000, 100)

So instead of dealing with 3072 entries, our models worked with 100 features, and this made our algorithms to run much faster.

1. **Analysis Process**

For the rest of this project, we decided to initially choose an unoptimized flexible model (that can be optimized) to build a classifier. Then we tried to optimize this model by tweaking the model parameters and adding some regularization method to get better performance metrics. For comparison purpose we chose the following models:

KNN,

Random Forest,

Gradient Boosted Decision Trees,

Artificial Neural Network,

Convolutional Neural Network.

We evaluated the performance of each of these initial models using the accuracy metrics.

For each of these cases we then optimized the initial models to get better metrics.

Finally, we compared all of these models considering the advantages and disadvantages and decided for the best model as our final classifier.

Since We did not have any restriction for having lower false-negative (e.g. Medical Diagnostic Classification) or lower false positive (e.g. Rejecting Junk Emails), therefore we decided to use “accuracy” metrics for final model selection and did not include Recall, Precision or False Positive Rate during our model training process.

1. **KNN**

KNN will find K nearest neighbors to a test point and classify this image on the base of class of these neighbors. Since some of the images in every category are very different from the other images in the same category (some frogs are in water background and the others are not), therefore this model which is based on distance and similarity could not compete with the other models we tried in this project. The initial accuracy we got in this model for out test data was: 0.911. We then tried to find the best number of K by using Cross Validation method and the result was K = 3. The best accuracy we got for our test data was 0.914. which is the lowest accuracy we got comparing to other methods.

1. **Random Forest**

Random Forest is one of the most widely used ensemble algorithms. It uses decision tree but combines bagging and random subspace learning. It randomly samples observations for each tree, and also randomly samples feature at each split.

For our initial random forest model with just 100 trees, we got accuracy of 0.922 for the test data. By using cross validation, we found that the best number of trees for our dataset was 300 and the best max\_depth (the length of the longest path from the tree root to a leaf) was 8. Applying these finding improved our accuracy to 0.924.

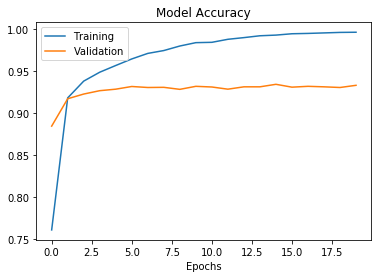
1. **Gradient Boosted Decision Trees**

Next, we decided to try Gradient Boosted algorithm. This is another approach for ensemble learning. This algorithm fits many decision tree models to our data and it reweights the data in every tree differently, this is done sequentially and the reweighing varies depends on the performance of each step. This decreases the bias and also the variance. For the initial Gradient Boosted model we got accuracy of 0.928 for the test data. Using Cross Validation gave us the best number of trees to be 300 and the best learning rate to be 0.1. Applying these parameters improved the model to have the accuracy of 0.933

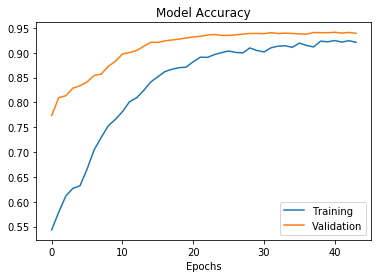
1. **Artificial Neural Network**

Artificial Neural Network is a network of layers. These layers can be thought of as passing data through a chain of functions. The layers simply apply different functions to a set of inputs and creates an output on the base of the inputs and the functions parameters. This process will be repeated many times to finally produce the best results. Using the initial Neural Net for our dataset here, we got the accuracy of 0.938 for our test data. Adding different kinds of regularization to his initial model not only increased the accuracy but also generalized our model which means it decreased the possibility of overfitting. The final accuracy we got after optimization for this model was: 0.941. The following accuracy graphs shows adding regularization guarantees that the model is not overfit to the training data (there is not a big gap between Training and Validation accuracy from one epoch to the next).

Initial Neural Net:



Optimized Neural Net:



1. **Convolutional Neural Network**

Convolutional Neural Network is an upgraded version of Artificial Neural Net. They work by adding two new constructs to feed-forward neural networks: convolutional layers and pooling layers. The convolutional layer is responsible for automatically extracting features from the images, and the pooling layer is used for dimension reduction. Instead of pre-processing the images with PCA for dimension reduction, they work by learning filters which extracts features from images automatically. Therefore, we had to reprocess our data at this step back to original none-PCAed format. Also, this algorithm gets the data in the four-dimension format and we had to change the format to the initial four-dimensional data. The result was as follows:

X shapes: (5000, 32, 32, 3) (5000, 32, 32, 3) (2000, 32, 32, 3)

Fitting the initial CNN with one convolutional and one pooling layer gave us the accuracy of 0.956 for our test data. As with regular Neural Net, here we also added the optimization and regularization methods which not only improved the accuracy but also helped with preventing the model to overfit to the training data. The final accuracy we got from this optimized model with all the different regularization methods applied was: 0.964, which is the best accuracy we got among all the other algorithms.

Here we can also look at the accuracy graphs, and again we can see adding the regularization prevented the model from overfitting to the train set:

Initial CNN:



Optimized CNN:



1. **Model Comparison and Conclusion**

As it can be seen in the above sections although optimizing in each step provided improvements to the initial model, but transitioning from one algorithm to the next also improved the accuracy metric. Here is the accuracy comparison table which shows the accuracy of all he models at a glance:

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Initial Accuracy** | **Optimized Accuracy** |
| KNN | 0.911 | 0.914 |
| Random Forest | 0.922 | 0.924 |
| Gradient Boosted | 0.928 | 0.933 |
| Neural Network | 0.938 | 0.941 |
| CNN | 0.956 | **0.965** |

With the final model accuracy to be the highest among all, we decided that the Convolutional Neural Network to be the most effective model both from accuracy perspective and also from lower bias/variance and overfitting concern. This result was expected since as we already know, CNNs are particularly useful for modeling datasets that have a grid-like structure, such as images where the pixels are laid out in a grid to form the picture. As a reference, here are the parameters for our final CNN model:

*Layers: Convolution with 16 filters, size 3, activation relu, strides 1, padding: same,*

*Pooling size 4, strides 2,*

*Convolution with 32 filters, size 3 activation relu, strides1, padding: same,*

*Pooling size 4, strides 3,*

*Fully connected with one node, activation sigmoid,*

*Dropout regularization with rate: .5*

*Optimizer: adam,*

*Loss: binary\_crossentropy,*

*Early Stopping with patience 3*

*Epochs 200,*

*Batch size 50*

**Convolutional Neural Network are notorious for being the best-performing models for image and video recognition problems, hence gave us the best result here.**