Image Classification Models on Satellite Data

By Jasmine Ng

I. Dataset

I used the "SAT-4 Airborne Dataset" provided by the National Agriculture Imagery Program and compiled by the Bay Area Environmental Research Institute/NASA Ames Research Center. The dataset consists of 500,000 satellite images of the Earth's surface. Each image is 28x28 pixels large. The images are in color and also include near infrared information. The goal is to build a classifier that will classify the images to either one of four labels: barren land, trees, grassland, or none. The dataset is divided into a training set consisting of 400,000 images and a set consisting of 100,000 images, both of which are labeled.

II. Classification Model I: Support Vector Machine

Before I designed my first classification model using support vector machines, I have transformed the dataset to be in grayscale so that I can compile the original four channels for each image into one channel. Then I have also reshaped the dataset to be two dimensional. This would allow my classifier to be more effective in training the dataset. To avoid overfitting, I have split my training dataset even further to have a validation set that is of .1 the size of the training dataset. Next, I ran a preliminary model with the split training set through the sklearn.svm package and a kernel-based SVM. Then I had used the params from the training dataset to predict the labels from the test set. The accuracy of the model with the test set came out to be 83%. I have found the accuracy to be satisfactory but this model took very long to run. Therefore, I would like to try using the linear SVM which would be much faster and could scale a lot better. Refer to Appendix A to see the classification report on this classification model.

III. Optimize Classification Model I: Support Vector Machine

The first attempt that I did to optimize the support vector machine classification model is to run through the same training dataset of grayscale values through the SVM using a linear kernel. However, I got a lower accuracy on the test set of around 54%. Refer to Appendix B to see the classification report on this classification model.

To further optimize the support vector machine, I first noticed that one of the disadvantages of support vector machines is that they are not scale invariant; therefore, I made the decision to normalize each feature to be between 0 and 1 before training the classifier. Next, I have flattened the four-dimensional array into a one-dimensional array of features. With the flattened features array, I looped over all 400,000 images to create features for each image and stacked the flattened features into a big feature matrix. The resulting feature matrix is shaped so that the rows represent the images and the columns contain the features with 400,000 images and 784 features. Next I have split the training set to have a validation test set that is .1 of its size. The training dataset has now the shape of (360000,784). To avoid overfitting, I wanted to reduce

the size of the features by linearly transforming the data so that most of the information in the data is contained within a small number of components. I did this by using sklearn's PCA built in function. Before running this through the model, I wanted to look at the distribution of labels in the train set to see if there is an even balance across all labels. I have noticed that there is a significant number of "none" images compared to the rest of the type of images. I have plotted the distribution of labels in the training set which can be referenced in Appendix C. In the case that this is an issue, I decided to change the metric from accuracy which is taking the proportion of correctly labeled images to be a Confusion Matrix--which is a breakdown of predictions into a table showing correct predictions and the types of correct predictions made. After running through the data through the SVM linear kernel, the model accuracy on the test set is .55 for weighted average. For more details on the performance of the model, reference Appendix D.

IV. Classification Model 2: Logistic Regression Using PyTorch

For the second model, I chose to do a logistic regression. The logistic regression is identical to a linear regression model and there are weight and bias matrices. The output is also done by using simple matrix operations which are (wx)+b. Using the transformed grayscale dataset, I transformed the dataset to a tensor form and concatenated with the labeled data. Next, I randomly picked 20% of the images for the validation set and randomly took a subset of each validation and training of size 100. Before feeding the resulting batch to the logistic regression model, I flattened each image tensor to a vector of size 784. The output for each image is a vector size of 4, with each element of the vector signifying the probability of a particular label. The predicted label for an image is the one with the highest probability.

Afterwards, I ran the resulting batch through the logistic regression model. By choosing the index of the element with the highest probability in each output row, I ran through the classifier of the prepared data. Next, I evaluated the metric and loss function to see how well the model is performing. To do this, I found the percentage of labels that were predicted correctly and this accuracy ends up being 40%. I have also used the cross entropy that is built into PyTorch. Unlike the accuracy metric I used prior, cross entropy is a continuous and differentiable function which makes it a good choice for a loss function. To optimize this metric, I ran through the model with various parameters in learning rate and batch size. I ended up choosing the optimal learning rate would be .05 since it gave a slightly higher accuracy for the correct label, indicating a lower loss when I have fitted the model over multiple epochs. Reference Appendix E to see a table that I have made to depict the accuracy of each trial I did with different learning rates.

In training the data, I built functions that would calculate the loss for a batch of data and the overall loss of the validation data. On the validation data, I got a 61% accuracy and a loss of 1.01. The accuracy is satisfactory since the accuracy is greater than 25%-- which is the chance that the model would correctly classify a label by randomly guessing. In terms of neural

networks, an epoch refers to one cycle through the full training dataset. In order to optimize the model, I have trained the data for more than one epoch, hoping for a better generalization when given a new set of batch data. In total, I trained about 18 epochs. In the figure below, I plotted a line graph to see the improvements through every epoch which can be seen in Appendix F.

It's quite clear from the above figure that the accuracy doesn't continue to increase as we train for more epochs. However, the model won't cross the threshold of 61% even after training for a long time. And for evaluating the model performance on the test data, I also obtained an accuracy of 61%. One possible reason for this is that the learning rate might be too high. It's possible that the model's parameters miss the optimal set of parameters that have the lowest loss. For further investigation, I can reduce the learning rate and training for a few more epochs to see if it helps.

The model may not have been as powerful as the SVM model because I have assumed that the result (probabilities) is a linear function of the pixel features from doing a matrix multiplication with the weight matrix and adding bias. However, there may not be a linear relationship between the two.

V. Classification Model 3: Stochastic Gradient Descent

I chose to use Stochastic Gradient Descent because it is a simple yet very efficient approach to learning linear classifiers under convex functions such as the linear SVM and Logistic Regression, the two models that I ran prior. The Stochastic Gradient Descent is advantageous since it is efficient. First, I had trained this model with a training set given in the SAT-4-full.mat. The accuracy of the model when used on the test set was around .52, more details in the performance of the model given in Appendix G.

VI. Optimize Classification Model 3: Stochastic Gradient Descent

In efforts to make this model perform better than the first time, I converted the training dataset to be a grayscale so that there would be only one channel. Then, I normalize each feature to be between 0 and 1 and flattened the data into a one-dimensional array of features. With the flattened features array, I looped over all 400,000 images to create features for each image and stacked the flattened features into a big feature matrix. The resulting feature matrix is shaped so that the rows represent the images and the columns contain the features with 400,000 images and 784 features. Next I have split the training set to have a validation test set that is .1 of its size. The training dataset has now the shape of (360000,784). To avoid overfitting, I wanted to reduce the size of the features by linearly transforming the data so that most of the information in the data is contained within a small number of components. I did this by using sklearn's PCA built in function. With the new training set, I ran the stochastic gradient descent model and then tried to predict the labels for the test data set with the fitted parameters from the training. This time

around, the accuracy has increased but not significantly. The accuracy obtained is around .56 and more details can be seen in Appendix H.

VII. Conclusion

In conclusion, I have decided to use the model with the highest accuracy. Surprisingly this came out to be the SVM classification model that is kernel based. The reason why this may be the strongest model that I have built is because the output in classification may not be linear with the input (the pixel features of the images).

VIII. Appendix

Appendix A: Classification report for SVM classification model (kernel based SVM)

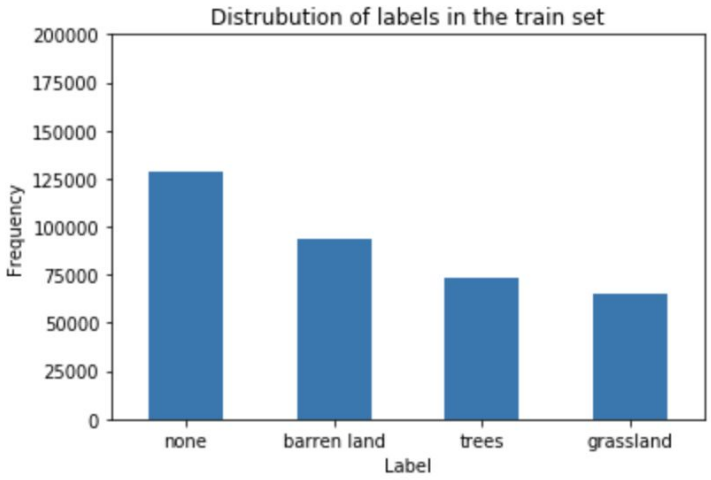
	precision	recall	f1-score	support
1	0.86	0.86	0.86	94073
2	0.86	0.87	0.87	73124
3	0.75	0.69	0.72	64596
4	0.83	0.86	0.85	128207
avg / total	0.83	0.83	0.83	360000

Appendix B: Classification report for the SVM classification model (linear kernel)

```
Accuracy: 0.5373194444444445
```

	precision	recall	f1-score	support
0	0.65	0.95	0.77	94073
1	0.50	0.82	0.62	73124
2	0.19	0.12	0.14	64596
3	0.58	0.28	0.38	128207
accuracy			0.54	360000
macro avg	0.48	0.54	0.48	360000
weighted avg	0.51	0.54	0.49	360000

Appendix C: Distribution of images in the training set



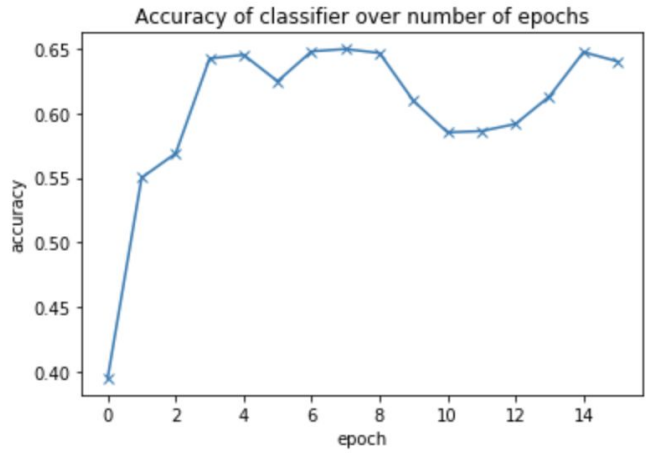
Appendix D: Classification report for the SVM Classification Model (Linear Kernel) with normalized and feature transformed dataset

```
Accuracy: [[10125    82    98    70]
 [   34  7511   302   299]
 [ 3087  2201   986   952]
 [ 3153  5250  2883  2967]]

              precision    recall  f1-score   support

           0       0.62      0.98      0.76     10375
           1       0.50      0.92      0.65      8146
           2       0.23      0.14      0.17      7226
           3       0.69      0.21      0.32     14253

    accuracy                           0.54     40000
   macro avg       0.51      0.56      0.47     40000
weighted avg       0.55      0.54      0.47     40000
```

Appendix E: Optimizing the accuracy of the model with different learning rates

Learning Rate	Loss/Accuracy	Threshold after multiple epochs	Loss/Accuracy on test set
.0001	1.4213/0.1926	.5223	1.4093/.5303
.001	1.1683/0.5255	.6051	1.0167/.6088
.005	1.3960/0.1759	.6278	0.9894/.6075
.01	0.9896/0.6057	.6489	0.9869/0.6242
.1	0.9865/0.6236	.5800	2.3166/0.3958
.05	2.3222/0.3951	.6471	1.0051/.6479

Appendix F: Accuracy of the Logistic Regression Model over various number of epochs



Appendix G: Classification report for the Stochastic Gradient Descent Model

Accuracy: 0.5177916666666667

	precision	recall	f1-score	support
0	0.66	0.96	0.78	94073
1	0.53	0.56	0.55	73124
2	0.15	0.15	0.15	64596
3	0.55	0.35	0.43	128207
accuracy			0.52	360000
macro avg	0.47	0.51	0.48	360000
weighted avg	0.51	0.52	0.50	360000

Appendix H: Classification report for the Stochastic Gradient Descent Model with normalized and feature transformed dataset

Accuracy: 0.5561

	precision	recall	f1-score	support
0	0.65	0.96	0.77	10375
1	0.52	0.78	0.62	8146
2	0.14	0.06	0.08	7226
3	0.58	0.39	0.47	14253
accuracy			0.56	40000
macro avg	0.47	0.55	0.49	40000
weighted avg	0.51	0.56	0.51	40000

Bibliography

1. S, Aakash N. “Image Classification Using Logistic Regression in PyTorch.” *Medium*, DataScienceNetwork(DSNet),13May2019, medium.com/dsnet/image-classification-using-logistic-regression-in-pytorch-ebb96cc9eb79.
2. Galarnyk, Michael. “PCA Using Python (Scikit-Learn).” *Medium*, Towards Data Science, 4 Nov. 2019, towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60.
3. Galarnyk, Michael. “PCA Using Python (Scikit-Learn).” *Medium*, Towards Data Science, 4 Nov. 2019, towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60.
4. “1.4. Support Vector Machines¶.” *Scikit*, scikit-learn.org/stable/modules/svm.html.