IST718 - Lab #9 - 6 June 2020

Author: John Fields

OVERVIEW

Using the Fashion MNIST dataset, can we use algorithms and compute to identify clothing items? Specifically, can we determine which algorithms and compute methodology provides us the most efficient approach for classifying simple fashion images?

Research Questions

1. What is the accuracy of each classification method?
2. What are the trade-offs of each approach?
3. What is the compute performance of each approach?

Summary Table for Performance

Model	Parameters	Time	Accuracy
TensorFlow and Keras Strides	3 epochs, 2 strides	48 sec.	90%
TensorFlow and Keras	10 epochs	62 sec.	89%
Linear Regression	1000 iterations	146 sec.	78%
Support Vector	n/a	899 sec.	89%
K Nearest Neighbor	n-neighbors = 3	147 sec.	89%
Random Forest	n-estimators = 100	84 sec.	89%
Naive Bayes	n/a	.41 sec.	59%
Auto Encoder	20 epochs	35 seconds	.03 validation loss

Summary Table for Trade-Offs of Each Model

Model	+	-
TensorFlow and Keras	Fast and good accuracy	Requires a GPU
Linear Regression	Fair speed	Poor accuracy
Support Vector	Good accuracy	Slow
K Nearest Neighbor	Good accuracy	Slow
Random Forest	Good accuracy and good speed	Not as fast as TensorFlow
Naive Bayes	Very Fast	Poorest accuracy
Auto Encoder	Fast and low validation loss	Different meausres (loss vs. accuracy)

Conclusion

TensorFlow/Keras with 2 strides provided the best balance of accuracy and speed. The algorithm was slightly better than Support Vector and K Nearest Neighbor with 89% accuracy but TensorFlow/Keras was faster with only 49 seconds for compute. Naive Bayes was the fastest at .41 seconds but also had the worst accuracy at 59%. One drawback to using TensorFlow/Keras is that it requires a GPU for optimal performance. TensorFlow/Keras without strides was able to converge on test results in only about 10 epochs. This was different from the ROC curve shown below for training data where convergence occurred at about 200 epochs.

The Auto Encoder was also accurate with a loss of .03 for validation data. The processing time of 35 seconds was also fast. The challenge with the Auto Encoder is that the measurement is not done for accuracy so you are not able to directly compare the results to the other methods.

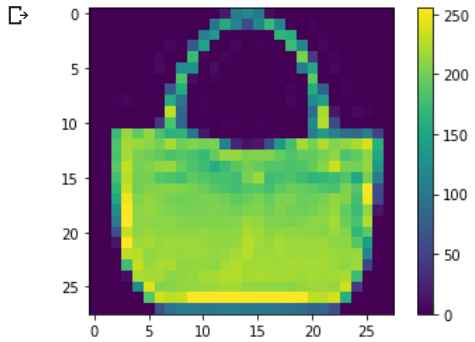
IMPORT DATA

DATA EXPLORATION

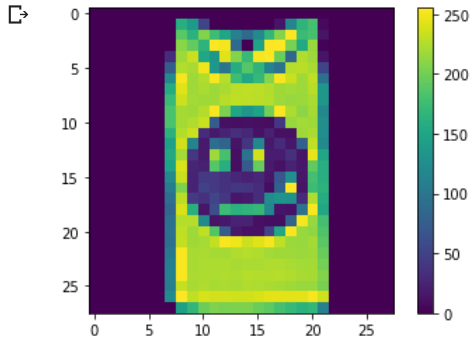
```
print(train_images.shape)
print(test_images.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

```
plt.figure()
plt.imshow(train_images[100])
plt.colorbar()
plt.grid(False)
plt.show()
```




```
plt.figure()
plt.imshow(train_images[101])
plt.colorbar()
plt.grid(False)
plt.show()
```

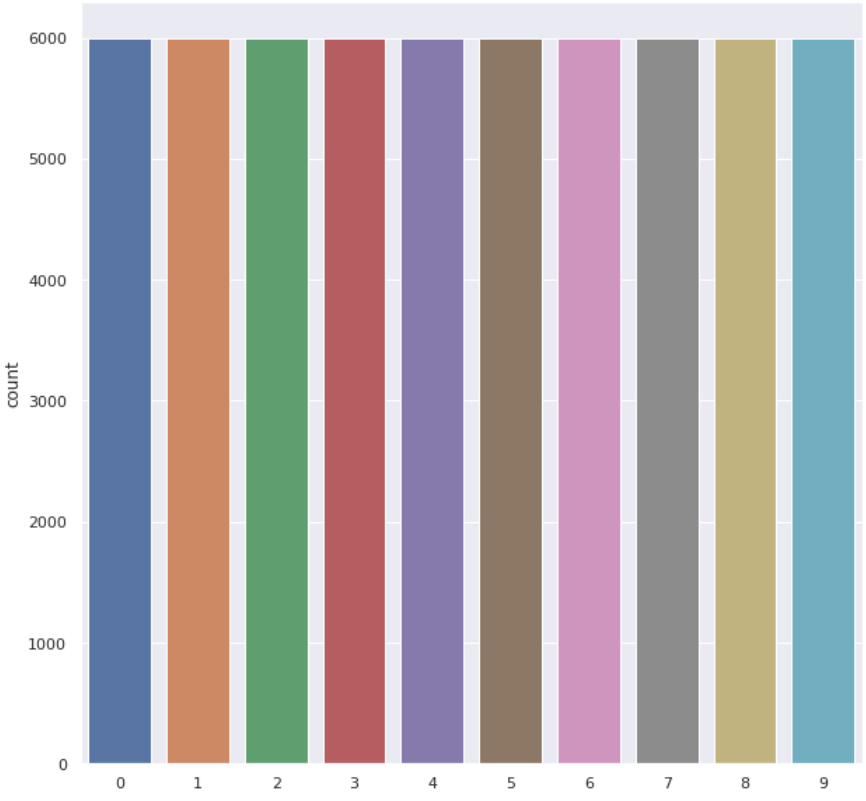



```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```


```
# Compare the number of examples per class in training
plt.figure(figsize = (10,10)) # Label Count
sns.set(style="darkgrid")
sns.countplot(train_labels)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6d738b128>
```



TENSOR FLOW AND KERAS MODEL

7 cells hidden

TENSOR FLOW AND KERAS RESULTS (300 EPOCHS)

Test accuracy: 0.8845999836921692

Elapsed Time (seconds): 1343.4857106208801

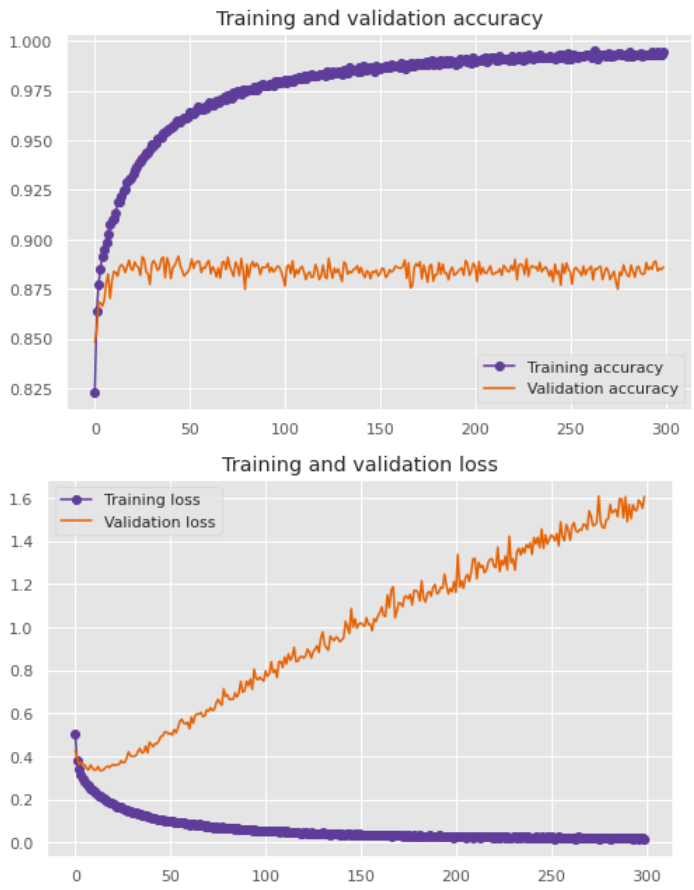
```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```


```
# INTERPRET
# BUILD FUNCTION FOR PLOTTING THE RESULTS OF THE MODEL
# From Week9_Keras_WalkThrough.ipynb

def plot_train_curve(history):
    colors = ['#e66101','#fdb863','#b2abd2','#5e3c99']
    accuracy = history.history['accuracy']
    val_accuracy = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(accuracy))
```

```
    with plt.style.context("ggplot"):
        plt.figure(figsize=(8, 8/1.618))
        plt.plot(epochs, accuracy, marker='o', c=colors[3], label='Training accuracy')
        plt.plot(epochs, val_accuracy, c=colors[0], label='Validation accuracy')
        plt.title('Training and validation accuracy')
        plt.legend()
        plt.figure(figsize=(8, 8/1.618))
        plt.plot(epochs, loss, marker='o', c=colors[3], label='Training loss')
        plt.plot(epochs, val_loss, c=colors[0], label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()
        plt.show()

plot_train_curve(history)
```



TENSOR FLOW AND KERAS MODEL - ADJUSTED STRIDE LENGTH

Adjusting the stride length provided an increase in the accuracy to 90.3%.

Code Source: https://www.kaggle.com/dansbecker/dropout-and-strides-for-larger-models

2 cells hidden

SKLEARN MODELS (LINEAR REGRESSION, SUPPORT VECTOR, K-NEAREST NEIGHBOR, RANDOM FOREST, NAIVE BAYES)

12 cells hidden

SKLEARN RESULTS

LINEAR REGRESSION

- Accuracy for Test: 0.793
- Elapsed Time (seconds): 136.29865431785583

SUPPORT VECTOR

- Accuracy for Test: 0.8921
- Elapsed Time (seconds): 898.5925350189209

K-NEAREST NEIGHBORS

- Accuracy for Test: 0.8949
- Elapsed Time (seconds): 146.99396920204163

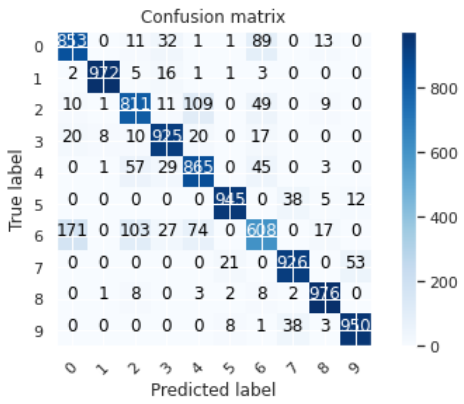
RANDOM FOREST

- Accuracy for Test: 0.8862
- Elapsed Time (seconds): 83.57019686698914

NAIVE BAYES

- Accuracy for Test: 0.5914
- Elapsed Time (seconds): 0.41146278381347656

```
# CONFUSION MATRIX FOR NAIVE BAYES
expected = test_y
predicted = preds
confusion_mtx = confusion_matrix(expected, predictions)
plot_confusion_matrix(confusion_mtx, classes = range(10))
```



AUTO ENCODER MODEL

Source: https://www.kaggle.com/nusretgencal/fashion-mnist-autoencoder-example

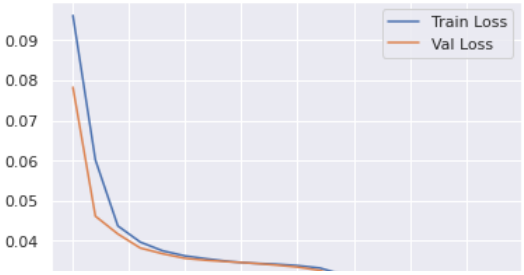
10 cells hidden

AUTO ENCODER RESULTS

Elapsed Time (seconds) for 20 epochs: 35.399916887283325

Plotting Losses

```
plt.plot(hist.history["loss"], label = "Train Loss")
plt.plot(hist.history["val_loss"], label = "Val Loss")
plt.legend()
plt.show()
```



Plotting the original and predicted image for the Auto Encoder Model

```
preds = autoencoder.predict(train_x)

from PIL import Image
f, ax = plt.subplots(1,10)
f.set_size_inches(80, 40)
for i in range(10):
    ax[i].imshow(train_x[i].reshape(28, 28))
plt.show()
f, ax = plt.subplots(1,10)
f.set_size_inches(80, 40)
for i in range(10):
    ax[i].imshow(preds[i].reshape(28, 28))
plt.show()
```