Luis Ferrufino

G#00997076

3/16/20

HW 2 (Part 1)

For my homework, I had to build three Keras ANN’s and three SVM’s, evaluate them, pick the best one, and evaluate it with noisy test data.

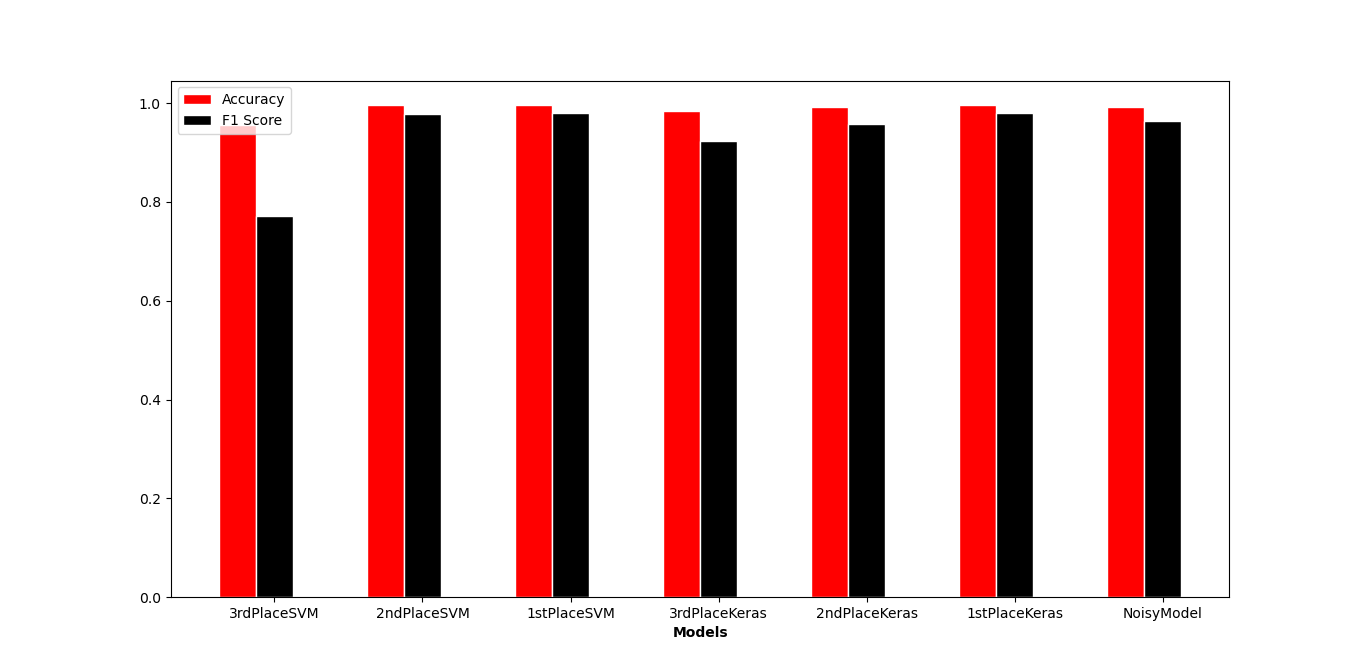
This project took me about thirty hours to complete, ten of which were spent trying install Tensorflow. The issue was that my computer’s processor doesn’t support AVX instructions, so I had to switch to another computer. In the end, I also had to downgrade from Python 3.8 to 3.7, and from Tensorflow 2.1 to 2.0.

To build my Keras models, I used a file called src/kerasBuilder.py. Every time I wanted to tune parameters or build a new model, I would modify the script or comment out some instructions. In this script, I had to load the training and testing data as numpy arrays, reshape them, cast them as floats, and normalize them by dividing each element by 255; for the training and testing labels, I had to convert them from a class vector to a binary class matrix format using to\_categorical(). After this preprocessing, the model is built, compiled, and fitted with the training data. Finally, an evaluation is done for the purpose of tuning parameters or deciding which model is better. Number of layers, activation functions, number of nodes, and number of epochs during model fitting were the parameters that I tuned. From worst to best, the models I saved were annThirdPlace.h5, annSecondPlace.h5 and MyBestModel\_g00997076.h5.

To build my SVM models, I used a file called src/svmBuilder. I tuned parameters and modified the file as I did with the previous file. Preprocessing was also the same, except I didn’t convert the labels into binary class matrices; I did however, concatenate the training and testing data (only used during the cross-validation for evaluation and parameter-tuning). The only parameter to tune here was the kernel function. Note that the currently uncommented portion of the code is for quickly building and saving the model, and that the commented portion was for evaluating models using cross-validation. Also note that I used sklearn’s SVC (Support Vector Classifier), which is an implementation of SVM. From worst to best the models I saved were svmThirdPlace.pkl, svmSecondPlace.pkl, svmFirstPlace.pkl.

src/addNoise.py was used to make a testing dataset with noise. I then went back to src/kerasBuilder.py to train my best model with it, and saved it as noisyModel.h5.

Now that all seven models had been built, I used src/performance.py to calculate the accuracy and F1 score for each class and average them, producing the average accuracy and average F1 score for each model. I then plugged this data into src/barPlot.py (which is my implementation of barPlotTemplate.py), and saved the chart as src/histogram.png:



noisyModel.h5’s performance came very close to that of MyBestModel\_g00997076.h5

The script I used to generate predictions on labels is called BestModel/tester.py. I did a test run on the given test data and saved the results in BestModel/results.txt. This is the content of BestModel/tester.py:

# Luis Ferrufino

# G#00997076

# 3/15/20

# tester.py

# HW2 (Part 1)

import keras

import numpy as np

from keras.models import load\_model

model = load\_model('./MyBestModel\_g00997076.h5')

#xTest = np.load('../src/MNIST\_X\_test\_1.npy')

#xTest = xTest.reshape(10000, 28\*28)

xTest = np.load('./Secret\_Test.npy')

xTest = xTest.reshape(xTest.size // (28\*28), 28\*28)

xTest = xTest.astype('float32')

xTest /= 255

predictions = model.predict\_classes(xTest, verbose=0)

f = open('./results.txt', 'w+')

for i in range(0, predictions.size):

f.write(str(predictions[i]))

if i < predictions.size - 1:

f.write('\n')

f.close()

BestModel/tester.py expects that the Secret\_Test.npy file will be in the BestModel folder. It will also write the predictions to a file called BestModel/results.txt

annThirdPlace.h5 had three layers in all, 10 nodes in the first, 1000 in the second, and 10 in the last; all layers used sigmoid as the activation function; it was fitted in 4 epochs. annSecondPlace.h5 had 10 layers: the first 9 nine layers had 50 nodes each, and used relu as the activation function; the last layer had 10 nodes and used softmax as the activation function; 3 epochs. MyBestModel.h5 had 3 layers: the first had 175 nodes and used softsign, the 3 hidden layers each had 300 nodes and used relu, and the last one had 10 nodes and used softmax; 5 epochs. I experimented with the activation functions and found that I got the best results with softmax in the last (i.e. output) layer. Also, the more epochs that were done in training, the lower the loss. According to the textbook (chapter 4.8.2), using relu (rectified linear units) in the hidden nodes helps the model overcome the vanishing gradient problem.

svmThirdPlace.pkl was an SVC using sigmoid as the kernel function. svmSecondPlace.pkl was an SVC using poly (degree 3) as the kernel function. svmFirstPlace.pkl was an SVC using rbf as the kernel function. There were other kernel functions I could have used (poly of degree 2 or 4, linear) but these took too long in the cross-validation. According to the textbook (chapter 4.9), even though choosing the kernel function depends on the characteristics of the data, rbf (radial basis function) is the most commonly used.