CPSC 340 Machine Learning Final Exam

Individual Portion

Spring 2020

Name: Zihan Liu
CS ID: zihanliu/s8y0b(old)
Student ID: 30647168

University of British Columbia
April 27, 2020

1 Introduction

The challenge is to perform classification of handwritten digits (i.e. 0–9) on the MNIST dataset: a collection of 60,000 handwritten digits and their corresponding labels. The goal is to train a suite of models outlined below on the training set (50000 examples) to be as accurate as possible when predicting on the test set (10000). Furthermore, the benchmarks to which implementations in this report are compared to are the best error for each classifier listed on MNIST website (`http://yann.lecun.com/exdb/mnist/`).

2 Methods

2.1 KNN

1. Model: KNN with PCA
2. Implementation detail: each image in the MNIST dataset is first deskewed so as to elimination any slant and misalignment. Then, an `sklearn GridSearchCV` is performed on different combinations of the following hyperparameters: *number of features* for PCA and k for KNN. The model is then trained on the optimal combination of said hyperparameters, and yields predictions.
3. Hyperparameters:
   - k for KNN: 3
   - *number of features (n_components)* for PCA: 65
   - *number of folds (cv)* for cross validation: 5
4. Justification of choice of hyperparameters: k and *n_components* are results from the optimization pipeline; *number of folds* used is arbitrary.

2.2 linear regression

1. Model: multi-class logistic regression with PCA, softmax loss and L2-regularization
2. Implementation detail: similar deskewing process is done as above. Then, an `sklearn GridSearchCV` is performed on: *number of features* for PCA and λ for L2-regularization strength, which is followed by standard training and prediction on best combination.
3. Hyperparameters:
   - *number of features (n_components)* for PCA: 65
   - λ for regularization strength: 0.1
   - *number of folds (cv)* for cross validation: 5
   - *number of gradient evaluations (maxEval)*: 500
4. Justification of choice of hyperparameters: *n_components* and λ are results from the optimization pipeline; *number of folds* used is arbitrary.

2.3 SVM

1. Model: multi-class SVM with polynomial kernel, PCA and L2-regularization
2. Implementation detail: similar deskewing process is done as above. Then, an `sklearn GridSearchCV` is performed on: *number of features* for PCA, λ for L2-regularization strength, and p for degree of polynomial kernel, which is followed by standard training and prediction on best combination.
3. Hyperparameters:
   - *number of features (n_components)* for PCA: 75

- λ for L2-regularization strength: 0.1
- p for degree of polynomial kernel: 4
- *coef* for degree of polynomial kernel: 1
- *number of folds (cv)* for cross validation: 5
- *number of gradient evaluations (maxEval)*: 500

4. Justification of choice of hyperparameters: *n_components*, λ, p, and *coef* are results from the optimization pipeline, *number of folds* used is arbitrary.

2.4 MLP

1. Model: MLP with L2-regularization and GD
2. Implementation detail: similar deskewing is done as above. Then, an `sklearn GridSearchCV` is performed on: *hidden_layer_sizes*, *learning rate (alpha)*, and λ, which is followed by standard training and prediction on best combination.
3. Hyperparameters:
   - *hidden_layer_sizes*: 128
   - *learning rate (alpha)*: 0.0001
   - λ for L2-regularization strength: 0.1
   - *number of folds (cv)* for cross validation: 3
   - *gradient iteration (max_iter)*: 500
4. Justification of choice of hyperparameters: *hidden_layer_sizes*, *learning rate (alpha)*, and λ are results from the `sklearn GridSearchCV`, while *gradient iteration (max_iter)* and *number of folds* are selected for runtime purposes.

2.5 CNN

1. Model: CNN using `Keras Sequential API`
2. Architecture: modified from `https://keras.io/examples/mnist\_cnn/`
   (a) layers: 2 conv layer, 1 fc layer, 1 max pooling layer, 2 dropout
      - input: 28*28*1
      - conv1: 32 3*3*1 filters, ReLu activated
      - conv2: 64 3*3*1 filters, ReLu activated
      - max pooling: 2*2 filter
      - fc1: 64*1
      - output: 10*1

   (b) regularization: dropout after max pooling and fc

   (c) optimization: SGD with momentum
3. Implementation detail: similar deskewing is done as above. Then, an `sklearn GridSearchCV` with `KerasClassfier` wrapper is performed on: (a)-(f) below, which is followed by standard training and prediction on best combination.
4. Hyperparameters:

(a) *batch_size*: 128

(b) *epochs*: 10

(c) *learn_rate*: 0.001

(d) *fc_neurons*: 64

(e) *conv1_neurons*: 32

(f) *conv2_neurons*: 64

(g) *conv_filter_sizes*: 3

(h) *maxpool_filter_size*: 2

(i) *momentum*: 0.9

(j) *dropout_rates*: 0.5

(k) *number of folds (cv)* for cross validation: 3

5. Justification of choice of hyperparameters: (a)-(f) are results from the `sklearn GridSearchCV`, while *conv_filter_sizes* and *maxpool_filter_size* are selected as the smallest to improve test error and to prevent overfitting. In addition, *momentum* and *dropout_rates* are selected from empirical evidence. Lastly, *number of folds* used is for runtime purposes, and the *weight initialization scheme* used for all layers is `Keras`' default: glorot_uniform.

3 Results

Model	Their Error (%)	Your Error (%)
KNN	0.52	1.600
linear regression	7.6	7.320
SVM	0.56	2.126
MLP	0.35	1.440
CNN	0.23	0.780

Note:

- all errors in the "Your Error" column are rounded to three decimal places.
- all errors in the "Their Error" column are collected from the MNIST website.

4 Discussion

Note: to avoid confusion, student implementation and implementation outlined on MNIST for each classifier above is referred to as "this implementation" and "MNIST implementation" respectively hereinafter.

In an attempt to justify why the results of this implementation are different from those of MNIST implementations, the following analysis is provided for each classifier.

1. K-Nearest-Neighbours (KNN):
   - absence of advanced preprocessing tools: steps mentioned on MNIST such as noise removal and image smoothing is not performed due to resource constraints.
   - absence of non-linear deformation analysis: such step is not performed due to similar reasons.
   - different distance function: some better performing MNIST implementations uses L3 norm while this implementation uses Euclidean distance since the specification of L3 norm is not readily accessible.
2. Linear regression (i.e. logistic regression):
   - use of PCA and regularization: this implementation uses standard L2-regularization and cross-validated PCA while it is unclear whether the same is done in MNIST implementations.
   - use of deskewing: this implementation uses deskewing as a preprocessing step while it is unclear whether the same is done in MNIST implementations.
3. Support Vector Machine (SVM):
   - smaller degree polynomial kernel: some better performing MNIST implementations uses polynomial kernels up to a degree of 9, which is not performed in student implementation due to time constraints.
   - training set sampling: it is likely that MNIST implementations use the training set in its entirety (i.e. 50,000 training examples) while this implementation only used 25,000 due to time constraints.
   - implementation details: compared to the similar degree-4 polynomial SVM MNIST implementation, this implementation is under-performing possibly due to suboptimal optimization method, etc.

   *Aside: experiment with naive multi-class linear SVM was also performed, but it yielded less ideal results (*4.920%*) than the polynomial-kernel SVM reported above, even though the former uses all training set. I suspect that this is due to the non-linear separability of MNIST dataset.*
4. Multi-Layer Perceptron (MLP):
   - lack of understanding of better optimization function: this implementation uses quadratic interpolation to optimize learning rate for GD, while MNIST implementation could have used more sophisticated methods on SGD.
   - sub-optimal weight initialization: this implementation uses traditional weight initialization of scaled $(1e-5)$ samples of standard normal while MNIST implementations might use more advanced techniques.
   - use of sigmoid as activation: this implementation might suffered from vanishing gradient problem.
   - naive regularization scheme: this implementation uses L2-regularization while more sophisticated methods such as dropout or early stopping could be used in MNIST implementations.

   *Aside: experiments with elastic distortion [Simard, 2003] was also performed, but it yielded slightly worse results (*1.49%*). Some potential reasons for this could be that a different implementation of elastic distortion is used in the original paper, or that the original paper used augmented on the original dataset before it was compiled into MNIST, etc. Also, in this implementation elastic distortion is only done on a sample of only 320 out of 50000 images due to runtime concerns, so I suspect the result could*

have been better if variations of more images are included in training. Please see `preprocessing.py` *for details.*

5. Convolutional Neural Network (CNN):
   - shallow architecture: this implementation adopts an architecture of three hidden layers (2 conv + 1 fc) due to runtime constraints while MNIST implementations use deeper networks.
   - lack of understanding of better optimization method: this implementation uses SGD as the optimization method, while MNIST implementations could have used those similar to `Keras`' implementation such as Adam, Adadelta which implements adaptive learning rate algorithms.
   - lack of understanding of better weight initialization method: this implementation uses `Keras`' standard glorot_uniform as the weight initializer, while MNIST implementatiosn could have used other methods due to the implications of weight distribution on the performance of CNN model.

   Aside: experiments with elastic distortion similar to above was also performed, however the results were also less ideal potentially due to similar reasons.