CAP 6618 Spring 2019
Christian Garbin
Module 3 Assignment

Objective of this assignment

The objective of this assignment is to "*Design and implement a complete solution to the problem of handwritten digit recognition, stated as follows: “given an image of a handwritten digit, produce the corresponding numerical value.”. Essentially, you will build, fine-tune, and evaluate a 10-class classifier which will employ the pixel values of the images as features.*"

The digits to be recognized are the ones from the MNIST dataset [1].

Files submitted

Two Jupyter notebooks are submitted as part of this assignment:

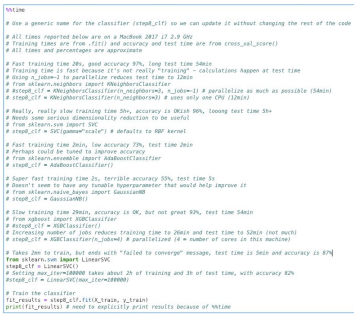
1. CAP6618_PP1_cgarbin_experiments.ipynb: based on the provided CAP6618_PP1.ipynb notebook, contains the experiments to choose a classifier. It also contains the answer to the assignment questions. Refer to the cells titled "Assignment question 4x", where "x" is one of the questions, from "a" to "g". The answers are also copied at the end of this report, for easier reference.
2. CAP6618_PP1_cgarbin_final.ipynb: contains the code using the chosen classifier, including the optimization steps.

Choosing a classifier

The first step for this report was to choose a classifier for the "design and implement" phase. SGDClassifier and RandomForestClassifier, both from scikit-learn, were excluded from the list because they were used as examples in the course material. All classifiers based on neural networks were also excluded as requested in the assignment details.

To choose a classifier, we evaluated a set of classifiers chosen after consulting [2], [3], [4], and [5].

The code used to choose the classifier is available in CAP6618_PP1_cgarbin_experiment.ipynb, in the "Step 8: Build, train, and evaluate a third classifier". Times were collected with Jupyter's `%%time` cell magic.



```
%%time

# Use a generic name for the classifier (step8_clf) so we can update it without changing the rest of the code

# All times reported below are on a MacBook 2017 i7 3.9 GHz
# Training times are from .fit() and accuracy and test time are from cross_val_score()
# All times and percentages are approximate

# Fast training time 20s, good accuracy 97%, long test time 50min
# Training time is fast because it's not really "training" — calculations happen at test time
# Using n_jobs=-1 to parallelize reduces test time to 12min
# from sklearn.neighbors import KNeighborsClassifier
# #step8_clf = KNeighborsClassifier(n_neighbors=4, n_jobs=-1) # parallelize as much as possible (54min)
# step8_clf = KNeighborsClassifier(n_neighbors=4) # uses only one CPU (12min)

# Really, really slow training time 5h+, accuracy is OK-ish 96%, looong test time 5h+
# Needs some serious dimensionality reduction to be useful
# from sklearn.svm import SVC
# step8_clf = SVC(gamma="scale") # defaults to RBF kernel

# Fast training time 2min, low accuracy 73%, test time 2min
# Perhaps could be tuned to improve accuracy
# from sklearn.ensemble import AdaBoostClassifier
# step8_clf = AdaBoostClassifier()

# Super fast training time 2s, terrible accuracy 55%, test time 1s
# Doesn't seem to have any tunable hyperparameter that would help improve it
# from sklearn.naive_bayes import GaussianNB
# step8_clf = GaussianNB()

# Slow training time 20min, accuracy is OK, but not great 93%, test time 54min
# from xgboost import XGBClassifier
# #step8_clf = XGBClassifier()
# Increasing number of jobs reduces training time to 26min and test time to 52min (not much)
# step8_clf = XGBClassifier(n_jobs=4) # parallelized (4 = number of cores in this machine)

# Takes 2mn to train, but ends with "failed to converge" message, test time is 5min and accuracy is 87%
from sklearn.svm import LinearSVC
step8_clf = LinearSVC()
# Setting max_iter=100000 takes about 2h of training and 3h of test time, with accuracy 82%
#step8_clf = LinearSVC(max_iter=100000)

# Train the classifier
(tl_results = step8_clf.fit(X_train, y_train)
print(tl_results) # need to explicitly print results because of %%time
```

Experiments to select a classifier. See the notebook for the code.

Table 1 shows the results of the experiments.

Classifier	Training time	Test time	Train+test time	Accuracy
`GaussianNB`	1 second	5 seconds	6 seconds	56%
`AdaBoostClassifier`	1 minute	2 minutes	3 minutes	73%
`LinearSVC`[1]	3 minutes	5 minutes	8 minutes	87%
`KNeighborsClassifier parallelized`[2]	19 seconds[3]	12 minutes	12 minutes	97%
`KNeighborsClassifier`	19 seconds	51 minutes	51 minutes	97%
`XGBClassifier parallelized`[4]	26 minutes	52 minutes	1:18 hour	93%
`XGBClassifier`	29 minutes	54 minutes	1:23 hour	93%
`SVC`[5]	6 hours	6 hours	12 hours	17%

Classifiers tried, ordered by test + training time. Note that times are in different units because of the disparity between the values.

Classifiers were evaluated for accuracy, training time and test time. Training time is the time taken by the `fit()` function. Test time is the time taken by `cross_val_score(cv=3,...)`. Accuracy is the average of the three folds from that function. All times are "wall time" from `%%time`, what the user experiences.

The goal was not to pick the classifier with the best accuracy, but the one with the best combination of training time, test time and accuracy. Test and training time are important factors because one of the objectives of this assignment is to learn how to optimize a classifier. Short feedback cycles speed up the optimization phase.

`KNeighborsClassifier parallelized` was chosen because of its high accuracy (highest of all) and reasonable training + test time.

[1] The LinearSVC classifier reported "failed to converge" with the default settings, recommending to increase the number of iterations. Increasing to 100,000 iterations (from the default 1,000) still resulted in "failed to converge" and increased training + test time to five hours. Dimensionality reduction could help to improve performance. To be investigated later.
[2] Two versions of KNeighborsClassifier, one without parallelization (first KNeighborsClassifier row) and another one with parallelization (n_jobs=-1, as many jobs as possible). The parallelized version was four times faster.
[3] "Training time" is misleading for a K-neighbors classifier. It does not have a training phase. All calculations are performed at test time, hence the large difference between training and test time for this type of classifier.
[4] Two versions of XGBClassifier, one without parallelization (first XGBClassifier row) and another one with parallelization (n_jobs=4, number of cores in the test machine). The parallelized version did not perform much better. [6] points to ways to solve that. Candidate to be investigated in another report.
[5] To be fair to the SVC classifier, it should have been tested with dimensionality reduction. MNIST is a sparse dataset (many unused pixels in each digit). Dimensionality reduction may speed up the training phase in this case. Candidate to be investigated in another report.

Optimizing the classifier

After the initial tests, with the results documented in the table above, the KNeighborsClassifier was optimzed.

Two important factors affect a KNN classifier:

- Scaling of parameters: a key concept in KNN classification is "distance", a measure of how connected a sample is to its neighbors. Since distance calculation involves the features of each sample (in the MNIST case, the value of each pixel), scaling may affect the the distance calculation and therefore clustering, which ultimately drives the classification.
- Hyperparameters: two hyperparameters are important in KNN classification, the number of neighbors to use in the distance calculation and what measure of "distance to use.

Experiments documented in CAP6618_P1_cgarbin_final.ipynb show that:

- Scaling reduces accuracy by a small, but significant margin. Scaling probably does not help in this case because the images are mostly comprised of pixels with zero value. It may have more of an effect if we first perform dimensionality reduction.
- The number of neighbors used in the calculation significantly affect the accuracy, with more neighbors reducing the accuracy.
- The Euclidean distance metric, the default value for KNeighborsClassifier, performs better than the alternative tried.

Please see the detailed results in the notebook.

Further areas to research

Some of the investigations point to the need to apply dimensionality reduction to the MNIST dataset.

More specifically, the SVC family of classifiers would certainly perform better and faster. LinearSVC is already fast (compared to others) and has reasonable accuracy. It would be interesting to investigate if dimensionality reduction would improve its accuracy even further, making it competitive with the KNN classifier, but much faster to run.

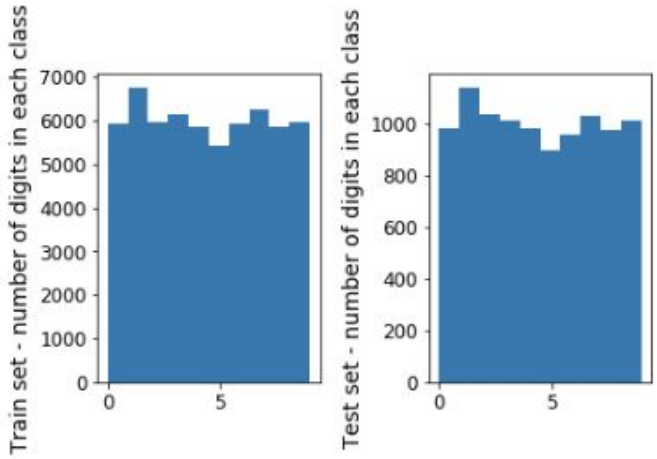
Answers to assignment questions

Answers to the questions in section 4 of the assignment are available inline in CAP6618_PP1_cgarbin_experiments.ipynb. Please refer to cells named "Assignment question 4x", where "x" is one of the letters in assignment 4 questions.

They are consolidated on this section for easier reference. In case of discrepancy between the notebook and this report, please consider the notebook the final reference.

Question 4a

Another useful data visualization is a histogram of the count of all classes. It checks if the classes are imbalanced, a source of headache for some classifiers. In the case of MNIST, they are balanced.



Question 4b

Stratified shuffling vs. random permutation: since the train and test datasets are large and balanced, with similar frequency for all classes, random permutation is enough in this case. Stratified sample would make a difference in imbalanced datasets.

Question 4c

This step uses a stochastic gradient descent classifier, i.e. it uses the gradient descent algorithm to optimize a loss function. At each step it will attempt to reduce the error (as defined by the loss function) by moving in the opposite direction of the gradient.

These are the most important parameters that affect this classifier:

1. Regularization/penalty: changes how the classifier generalizes (reduces overfitting). The first parameter to play with if the classifier performs well in training/validation, but performs badly in testing/prediction.
2. Learning rate: affects the speed of the convergence, or in some cases, if the classifier converges at all. It can be a defined value or an adaptive rate, one that will change when the classifier is not improving.
3. The number of iterations: how many times the classifier will go over the training data. Prevents the classifier from spending lots of computing resources without commensurable improvements in results.

Other parameters that may be of interest are:

1. The loss function: most of the time the standard RMSE (root mean square error) will be enough, but for some datasets other loss functions may be more useful.
2. The regularization/penalty constant: adjusts how sensitive the classifier is to the regularization terms. Most of the time the small default value is enough, but it may need to be increased in case of severe overfitting.

3. Early stopping: stops the optimization process if the classifier is not improving after a certain number of iterations. It is particularly useful for computation-intensive classifiers.

Question 4d

Cross-validation score parameters:

1. 1. The classifier to be evaluated.
2. 1. The training dataset.
3. 1. The training labels (matching the training dataset samples).
4. 1. The number of folds to use for splitting. It will fit (train) the classifier in n-1 of the folds and validation (test) on the remaining folder. This is repeated for all combination of training and validation folders.
5. 1. The metric ("scoring") to apply to the classifier. Since we are working with a multilabel classification problem on a well-balanced dataset, "accuracy" is an adequate metric.

Question 4e

This section uses a `RandomForestClassifier`. It is a set of decision tree classifiers, combined into one output by averaging the output of each decision tree.

Parameters that significantly affect this classifier are:

1. Number of estimators: the number of trees in the estimator. Too few of them may result in low variance, but high bias (underfitting). Too many of them may increase variance, but reduce bias (overfitting).
2. The maximum depth can help choose between variance and bias.

Question 4f

This code scales the training set and trains the classifier on that data.

It allows us to compare the results of the classifier before we scale the training data (cells above) and after we scale it.

Question 4g

Scaling the training data or not affects some classifiers and does not affect others.

Classifiers that calculate formulas that use the actual feature data (the actual value of each feature) are affected by scaling because some features (usually the ones with high values) can dominate features with smaller values. This may unduly give more weight to a feature simply because it has a higher absolute value.

Some classifiers are not affected by scaling. These are the ones that do not perform calculations using the feature values.

Random forester is one such classifier. It's not affect much by scaling. In this case in paticular, scaling the features reduced overall accuracy.

Sources

[1] LeCun et al., The MNIST Database, http://yann.lecun.com/exdb/mnist/, accessed on February 7th, 2019.

[2] M. Thoma, Comparing Classifiers, https://martin-thoma.com/comparing-classifiers/, accessed on February 7th, 2019.

[3] scikit-learn classifier comparison, https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html, accessed on February 7th, 2019.

[4] J. Brownlee, How to Develop Your First XGBoost Model in Python with scikit-learn, https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/, accessed on February 7th, 2019.

[5] LeCun et al., Learning Algorithms For Classification: a Comparison on Handwritten Digit Recognition, accessed on February 7th, 2019.

[6] J. Brownlee, How to Best Tune Multithreading Support for XGBoost in Python, https://machinelearningmastery.com/best-tune-multithreading-support-xgboost-python/, accessed on February 9th, 2019.

[7] Prasath et al. "Distance and Similarity Measures Effect on the Performance of K-Nearest Neighbor Classifier – A Review"