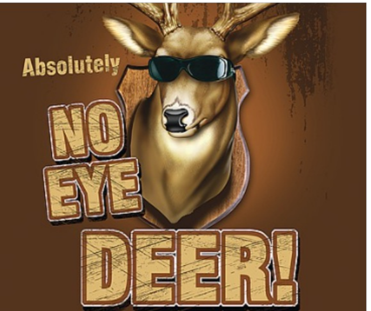
Classification of Handwritten Chinese Number Characters

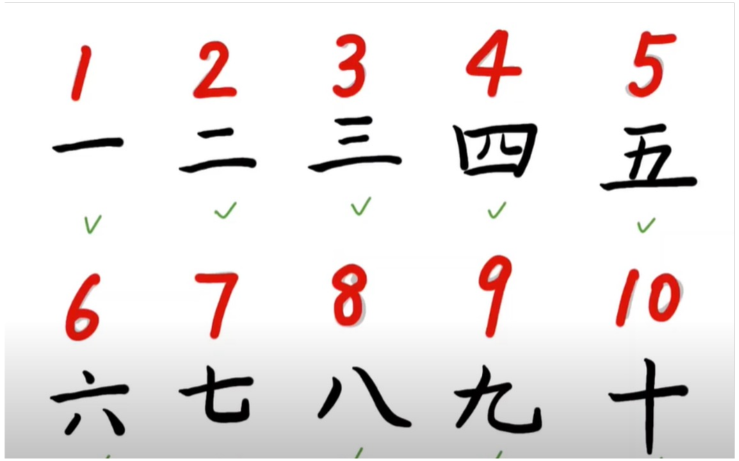
Team: No Eye Deer
Beibei Du, Jeffrey Feng, Luning Yang, Jiayi Zhao, Tian Zhong, Huaning Liu



Motivation

With pattern recognition algorithms, we can easily combine computer vision to relieve information from bills, printed documents, and writings.

For this prediction task, we want to be able to compare different machine learning algorithm on their performance to classify the handwritten number characters in Chinese.



Literature Review

- Source 1: Chinese Character Recognition
- This paper explores general methods of character recognition and suggests a formula that fits more closely with Chinese Handwritings.
- Why Interesting: Rather than following traditional pixel-splitting techniques to address image processing problems, it develops a new formula as approach that might be more general for Chinese characters
- Why select this (relate to our study): Although we’re not looking in-depth about the formula things, we share the same topics and thus hope to find inspirations from their approaches.
- How is this relating to other papers: The proposed algorithms are all formula-based.

Literature Review

- Source 2: Dimension Reduction for Classification Cases
- This paper deals with several dimension reduction algorithms to help accelerate and optimize the dimension reduction and feature selection process, including filter algorithms, hybrid algorithm, etc. And attempts to formulate a integrated intelligent feature selection system.
- Why Interesting: It attempts to actually automize the dimension reduction process
- Why select this (relate to our study): This might be useful if we have too much dimensions when building the model, which might lead to long running time.
- How is this relating to other papers: Compared with other papers that attempts to build classification models for labels or characters, this paper focuses on the platform of automating dimension reduction.

Literature Review

- Source 3: NLP & Classification
- This paper first reviews some recent findings about supported entity types in NLP, then proposes several algorithms to tackle the Named Entity Recognition and Classification (NERC) tasks.
- Why Interesting: I really like the EDA part, looking at the character statistics from many perspectives
- Why select this (relate to our study): Although the NERC topic in NLP is too abstract and in-depth compared with our project, I think they proposed good approach of a classification process, including feature engineering and, modeling.
- How is this relating to other papers: Compared with other papers, this, rather than developed a formula or algorithm, parallelly proposed and lists the evaluation of several models, and presents the comparing result, which is relatively a traditional ML process.

Data

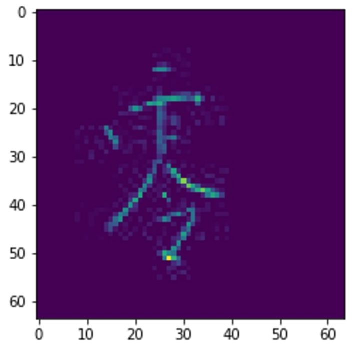
MNIST is a large database of handwritten digits that is commonly used for training various image processing systems. We found the Chinese MNIST dataset on Kaggle, which includes 72-MB 15000 compressed images. Each image shows one character from a set of 15 characters.

The data can be found here: https://www.kaggle.com/gpreda/chinese-mnist?select=data

Cleaning/EDA

The distinct labels and their representing Chinese characters can be found in the table below on the left, the 15000 instances of characters, along with the array representation of the image is shown on the right.

Pixel: 64 * 64



	value	character	code
0	0	零	1
1	1	一	2
2	2	二	3
3	3	三	4
4	4	四	5
5	5	五	6
6	6	六	7
7	7	七	8
8	8	八	9
9	9	九	10
10	10	十	11
11	100	百	12
12	1000	千	13
13	10000	万	14
14	100000000	亿	15

Experiment: Binary Classification

Sklearn: logistic regression

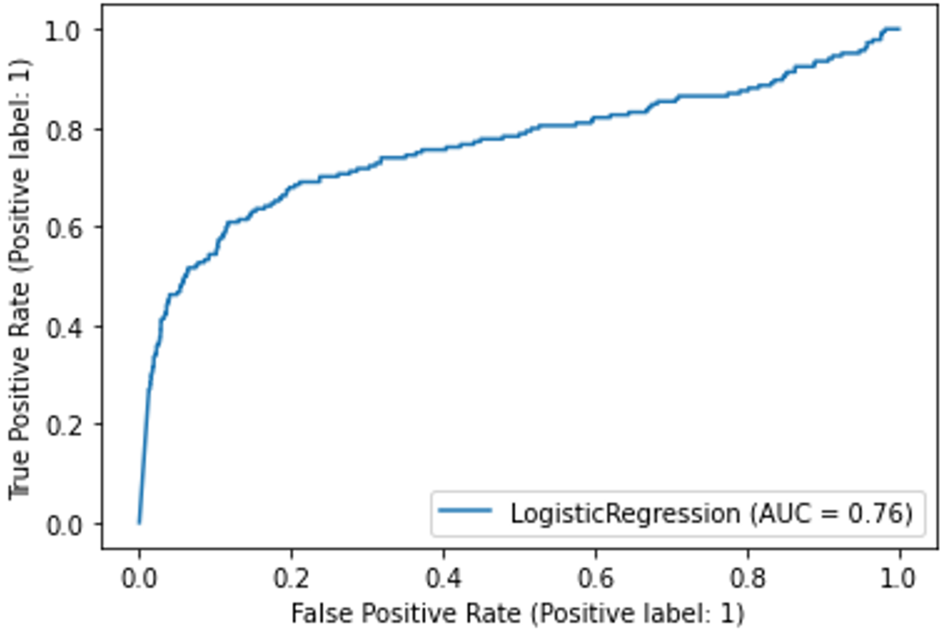
Task: identify all the images with label "1"

Split: 80% train, 20% test

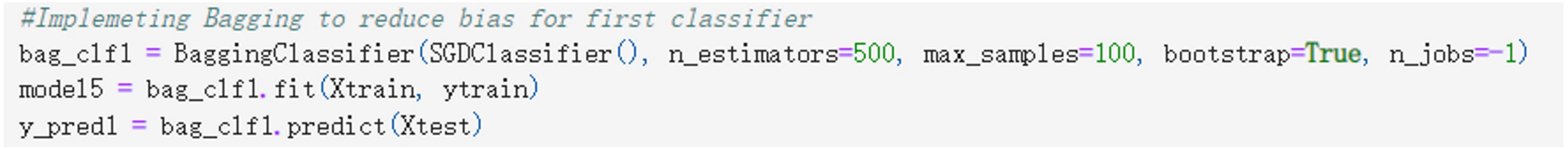
Performance with penalty:

- train accuracy 100%
- test accuracy 90%

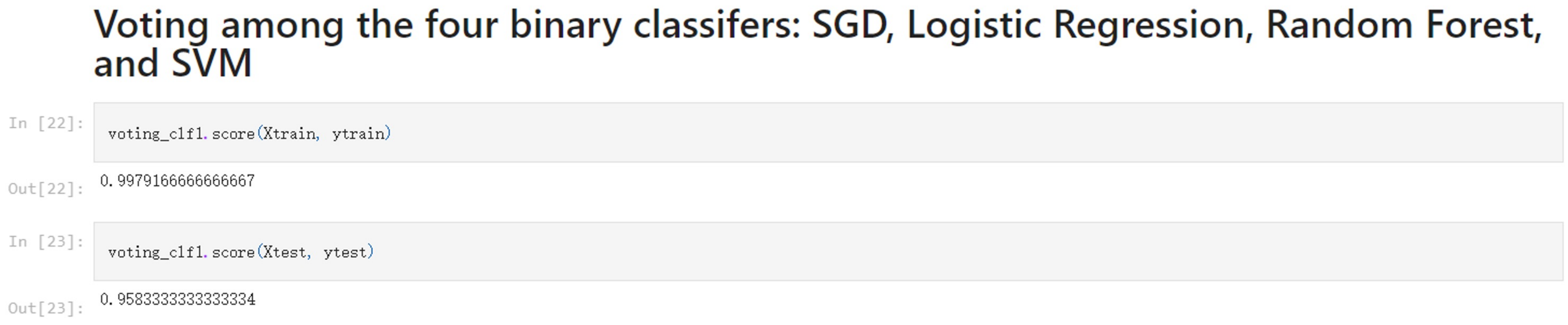
This is not the accuracy for the multi-class prediction!



Bagging Binary Classifiers



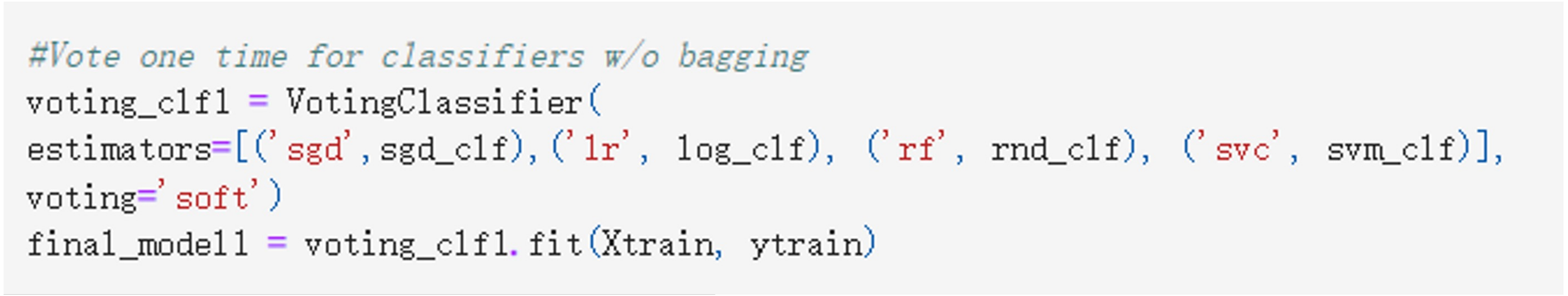
```python
#Implemeting Bagging to reduce bias for first classifier
bag_clf1 = BaggingClassifier(SGDClassifier(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1)
model5 = bag_clf1.fit(Xtrain, ytrain)
y_pred1 = bag_clf1.predict(Xtest)
```



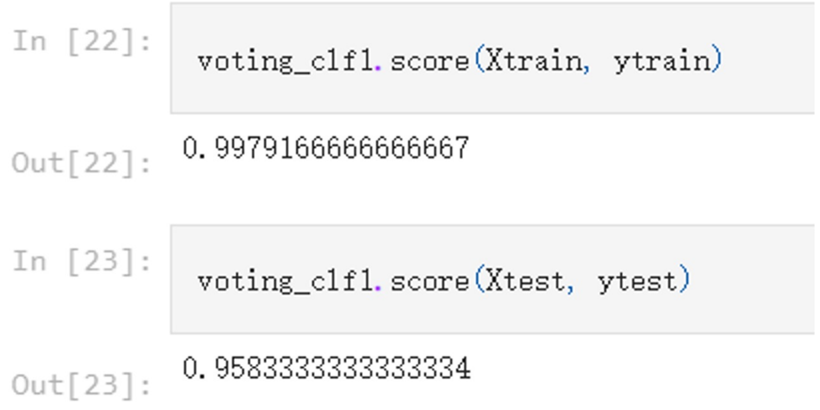
Voting among the four binary classifers: SGD, Logistic Regression, Random Forest, and SVM

```
In [22]: voting_clf1.score(Xtrain, ytrain)

Out[22]: 0.9791666666666667

In [23]: voting_clf1.score(Xtest, ytest)

Out[23]: 0.9583333333333334
```

Voting Binary Classifiers



```
#Vote one time for classifiers w/o bagging
voting_clf1 = VotingClassifier(
estimators=[(’sgd’,sgd_clf),(’lr’, log_clf), (’rf’, rnd_clf), (’svc’, svm_clf)],
voting=’soft’)
final_model1 = voting_clf1.fit(Xtrain, ytrain)
```

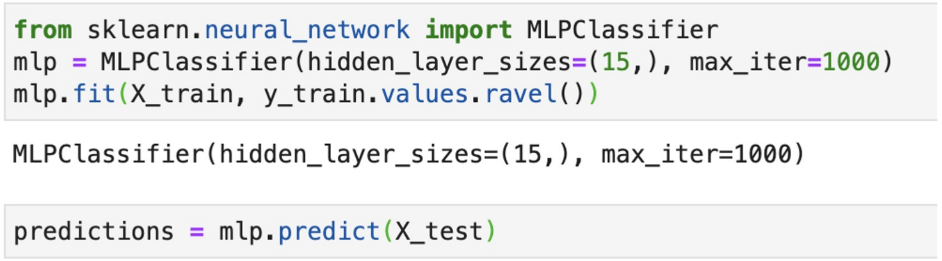


```
In [22]:  voting_clf1.score(Xtrain, ytrain)

Out[22]:  0.9979166666666667

In [23]:  voting_clf1.score(Xtest, ytest)

Out[23]:  0.9583333333333334
```

Artificial Neural Network(Multilayer Perceptron) with sklearn

Multi-layer perceptrons, or artificial neural networks, are a combination of multiple neurons connected in the form a network. Unlike logistic regression, an artificial neural network has an input layer, one or more hidden layers, and an output layer.

We use Python's Scikit-Learn library to create our neural network that performs this classification task. Given a set of features X and a target y, it can learn a nonlinear function approximation for classification or regression.



```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(15,), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
```

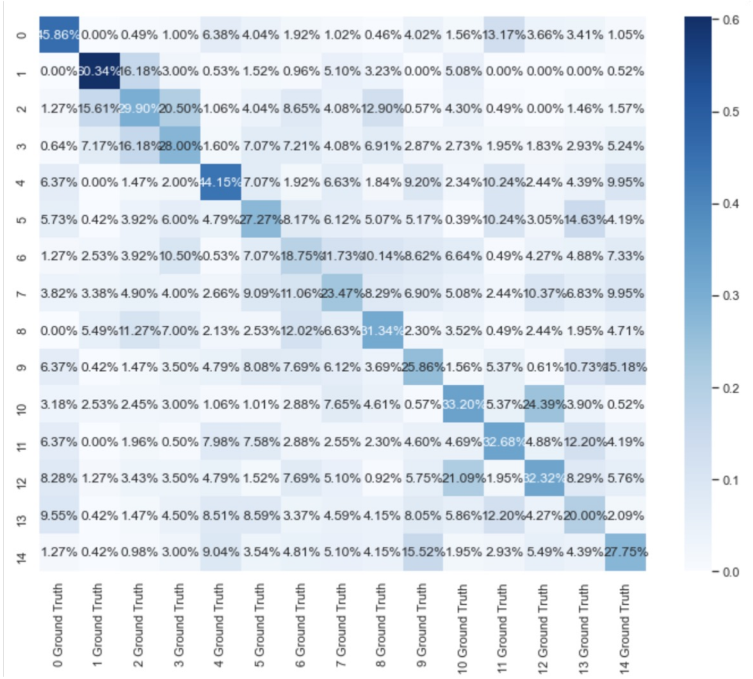
```
MLPClassifier(hidden_layer_sizes=(15,), max_iter=1000)
```

```
predictions = mlp.predict(X_test)
```

MLP with sklearn

```python
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

You can see the confusion matrix that every row is ground truth and every column is prediction



	0 Ground Truth	1 Ground Truth	2 Ground Truth	3 Ground Truth	4 Ground Truth	5 Ground Truth	6 Ground Truth	7 Ground Truth	8 Ground Truth	9 Ground Truth	10 Ground Truth	11 Ground Truth	12 Ground Truth	13 Ground Truth	14 Ground Truth
0	45.86%	0.00%	0.49%	1.00%	6.38%	4.04%	1.92%	1.02%	0.46%	4.02%	1.56%	13.17%	3.66%	3.41%	1.05%
1	0.00%	60.34%	16.18%	3.00%	0.53%	1.52%	0.96%	5.10%	3.23%	0.00%	5.08%	0.00%	0.00%	0.00%	0.52%
2	1.27%	15.61%	29.90%	20.50%	1.06%	4.04%	8.65%	4.08%	12.90%	0.57%	4.30%	0.49%	0.00%	1.46%	1.57%
3	0.64%	7.17%	16.18%	28.00%	1.60%	7.07%	7.21%	4.08%	6.91%	2.87%	2.73%	1.95%	1.83%	2.93%	5.24%
4	6.37%	0.00%	1.47%	2.00%	44.15%	7.07%	1.92%	6.63%	1.84%	9.20%	2.34%	10.24%	2.44%	4.39%	9.95%
5	5.73%	0.42%	3.92%	6.00%	4.79%	27.27%	8.17%	6.12%	5.07%	5.17%	0.39%	10.24%	3.05%	14.63%	4.19%
6	1.27%	2.53%	3.92%	10.50%	0.53%	7.07%	18.75%	11.73%	10.14%	8.62%	6.64%	0.49%	4.27%	4.88%	7.33%
7	3.82%	3.38%	4.90%	4.00%	2.66%	9.09%	11.06%	23.47%	8.29%	6.90%	5.08%	2.44%	10.37%	6.83%	9.95%
8	0.00%	5.49%	11.27%	7.00%	2.13%	2.53%	12.02%	6.63%	31.34%	2.30%	3.52%	0.49%	2.44%	1.95%	4.71%
9	6.37%	0.42%	1.47%	3.50%	4.79%	8.08%	7.69%	6.12%	3.69%	25.86%	1.56%	5.37%	0.61%	10.73%	15.18%
10	3.18%	2.53%	2.45%	3.00%	1.06%	1.01%	2.88%	7.65%	4.61%	0.57%	33.20%	5.37%	24.39%	3.90%	0.52%
11	6.37%	0.00%	1.96%	0.50%	7.98%	7.58%	2.88%	2.55%	2.30%	4.60%	4.69%	32.68%	4.88%	12.20%	4.19%
12	8.28%	1.27%	3.43%	3.50%	4.79%	1.52%	7.69%	5.10%	0.92%	5.75%	21.09%	1.95%	32.32%	8.29%	5.76%
13	9.55%	0.42%	1.47%	4.50%	8.51%	8.59%	3.37%	4.59%	4.15%	8.05%	5.86%	12.20%	4.27%	20.00%	2.09%
14	1.27%	0.42%	0.98%	3.00%	9.04%	3.54%	4.81%	5.10%	4.15%	15.52%	1.95%	2.93%	5.49%	4.39%	27.75%

```
              precision    recall  f1-score   support

           1       0.52      0.47      0.49       215
           2       0.58      0.63      0.61       190
           3       0.29      0.23      0.26       194
           4       0.28      0.27      0.27       202
           5       0.34      0.34      0.34       189
           6       0.25      0.23      0.24       231
           7       0.30      0.31      0.30       217
           8       0.19      0.20      0.20       197
           9       0.34      0.32      0.33       187
          10       0.23      0.25      0.24       195
          11       0.32      0.39      0.35       192
          12       0.26      0.28      0.27       201
          13       0.33      0.31      0.32       197
          14       0.23      0.24      0.24       198
          15       0.32      0.30      0.31       195

    accuracy                           0.32      3000
   macro avg       0.32      0.32      0.32      3000
weighted avg       0.32      0.32      0.32      3000
```

Also, the f1 score of 0.32 is quit bad, given the fact that we had 3000 instances to train.

Baseline model: Logistic regression

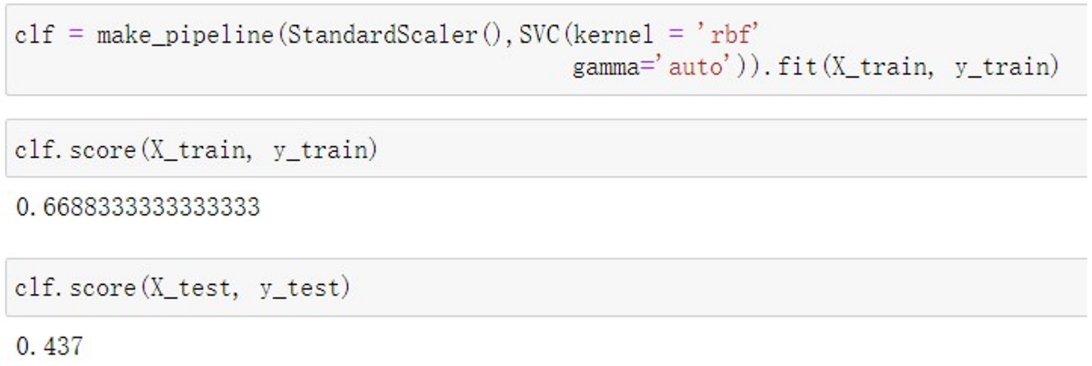
Sklearn logistic regression has a multinomial classification predictor.

Train accuracy: 67%, test accuracy 36.2%

Take away: try grid search, different algorithms

SVM

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.



```
clf = make_pipeline(StandardScaler(), SVC(kernel = 'rbf'
                                          gamma='auto')).fit(X_train, y_train)
```

```
clf.score(X_train, y_train)
```

```
0.6688333333333333
```

```
clf.score(X_test, y_test)
```

```
0.437
```

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

Radial Basis Function (RBF) Kernel

Random Forest

Random forest classifier implements classification based on several decision trees (`n_estimators`). Each decision tree gives a prediction outcome, and the final result of the random forest depends on the outcome that occurs the most times among the decision trees.

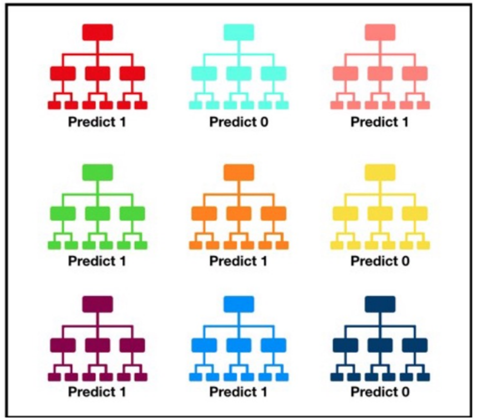
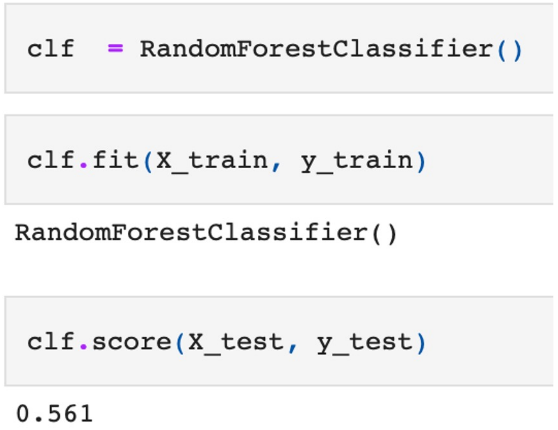


Image from: https://towardsdatascience.com/understanding-random-forest-58381e0602d2



```
clf  = RandomForestClassifier()
```

```
clf.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

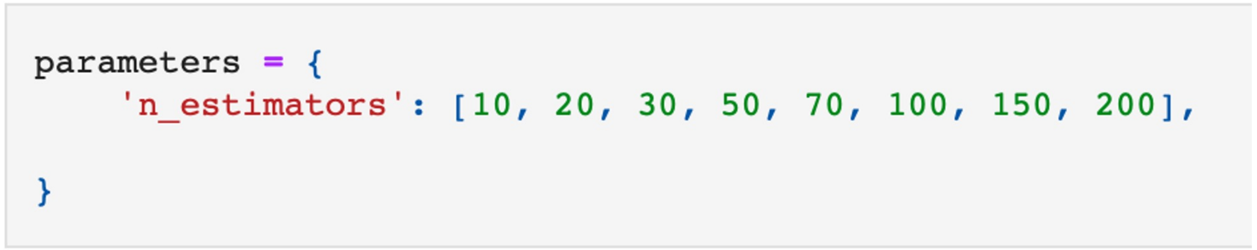
```
clf.score(X_test, y_test)
```

```
0.561
```

Simply running the algorithm, we get an accuracy of 0.561

Grid Search

How can we decide how many decision trees we should use for a given random forest model? Fit many RF models with different `n_estimator` and compare their performance?

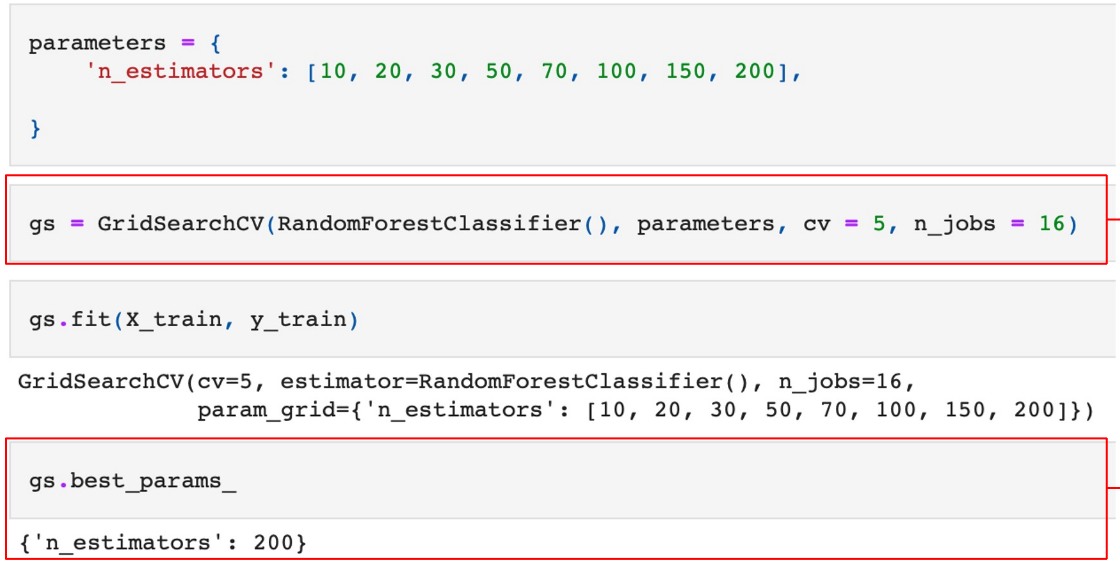


```
parameters = {
    'n_estimators': [10, 20, 30, 50, 70, 100, 150, 200],

}
```

For example, if there are 8 possible choices for the `n_estimators`, how do you compare them?

Grid Search

One way is to use the method GridSearchCV



```
parameters = {
    'n_estimators': [10, 20, 30, 50, 70, 100, 150, 200],

}
```

```
gs = GridSearchCV(RandomForestClassifier(), parameters, cv = 5, n_jobs = 16)
```

```
gs.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=16,
             param_grid={'n_estimators': [10, 20, 30, 50, 70, 100, 150, 200]})
```

```
gs.best_params_
```

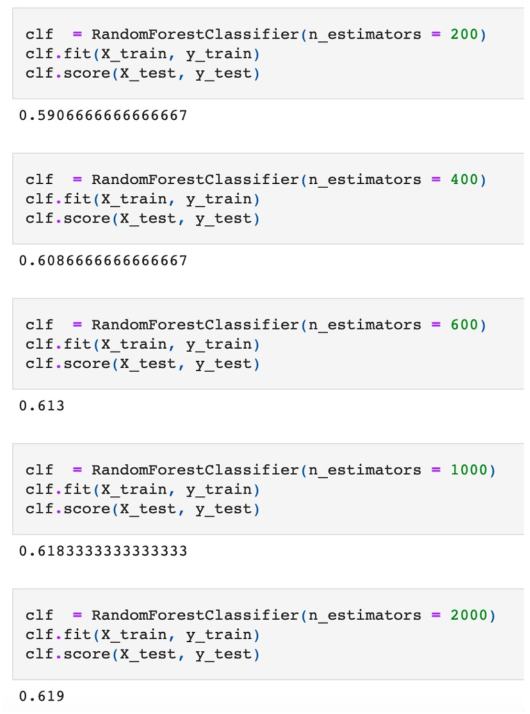
```
{'n_estimators': 200}
```

create a model per combination of the parameters and assess the performance based on K-fold validation

The model that has the best performance is when `n_estmators` = 200

Inference on the result

It is interesting to note that the best `n_estimators` is the largest one among our choices. Does that mean the test accuracy will increase as `n_estimators` increases?



```
clf  = RandomForestClassifier(n_estimators = 200)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.5906666666666667
```

```
clf  = RandomForestClassifier(n_estimators = 400)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.6086666666666667
```

```
clf  = RandomForestClassifier(n_estimators = 600)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.613
```

```
clf  = RandomForestClassifier(n_estimators = 1000)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.6183333333333333
```

```
clf  = RandomForestClassifier(n_estimators = 2000)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.619
```

This is somehow true when we increase the `n_estimators` from 200, 400, 600, till 1000. However, the test accuracy doesn't increase a lot when we double our `n_estimators` from 1000 to 2000.

This is true probably because our model involves 4095 pixel values as features. A high number of features demands a high model complexity.

ADABoost Classifier with scikit-learn

- A basic boosting algorithm that repetitively fits the classifier based on the mistakenly predicted cases last time
- This, by my opinion, is quite a baseline model in gradient boosting algorithm, as the original correct label might be predicted wrong when optimizing the original wrong ones.
- After tuning by GridSearch, we choose the n_estimators to be 300, and learning rate to be 1, with a base estimator of Random Forest.
- The train score is reported to be 1, with a test accuracy of 57.3%.
- This is even lower than previous tuned random forest model, which indicates a possible overfitting case

XGBoost Classifier

- This is a high efficient gradient boosting algorithm to boost tree algorithms
- Formula behind:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\overset{\text{instances mapped to leaf}_j}{\boxed{\left(\sum_{i\in I_j} g_i\right)^2}}}{\sum_{i\in I_j} h_i + \lambda} + \gamma T.$$

The tree learner structure q scoring function

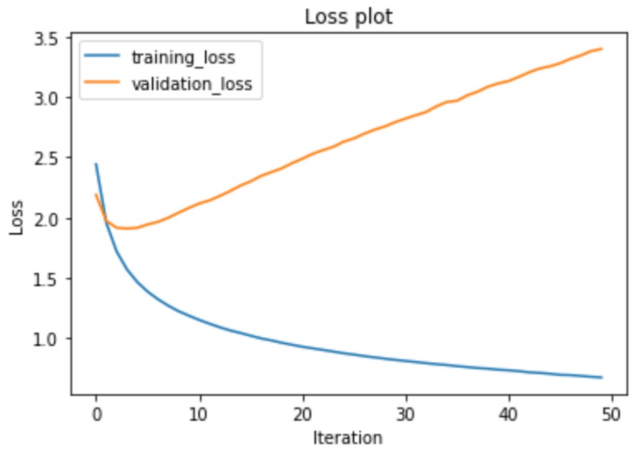
- The train score is reported to be 0.99775, the test accuracy is reported to be 0.627
- Take-away: This algorithm performs better, as it no longer only focuses on the “error” ones in the previous training.

LightGBM Classifier

- LGBM is a gradient boosting framework based on tree algorithms developed by Microsoft. Compared with previous two Gradient Boosting Algorithms, it takes faster running time and generally strong performance, but being more sensitive to overfitting.
- To avoid overfitting, we try to restrict the max_depth and learning rate parameter to be 12 and 0.01 to limit the growth of trees. Then, grid search another important parameter -- num_leaves, and set it to be 41.
- The train accuracy is reported to be 100%, the test accuracy is reported to be 71.8%.
- Still being in the risk of overfitting, but the performance has been good enough for LGBM.
- Balanced Error Rate: 29.4%

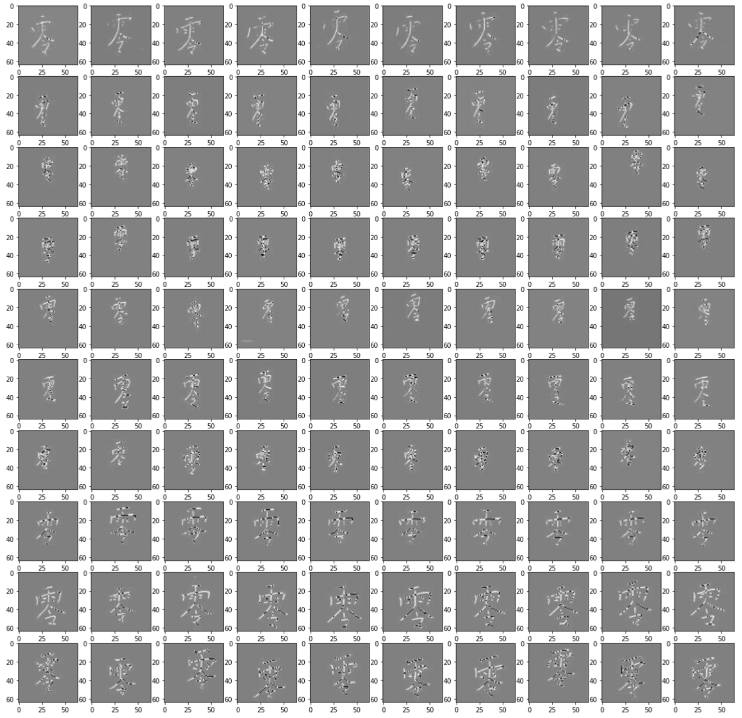
Multi-Layer Perceptron Classifier with PyTorch

- This is a extension of previous MLP model, in which I rewrite the algorithm with PyTorch and detect if the accuracy changes
- After several attempts, we settle the number of epochs to be 50, learning rate to be 0.01 and batch size to be 64.
- We take the criterion as cross entropy loss and optimizer as Adam.
- Train Accuracy: 79.52%, Test Accuracy: 35.2%
- Take-away: Issue of Overfitting
- Note: Validation is actually test set here
- Note: SGD causes a disaster



Problem with conventional supervised ML algorithm

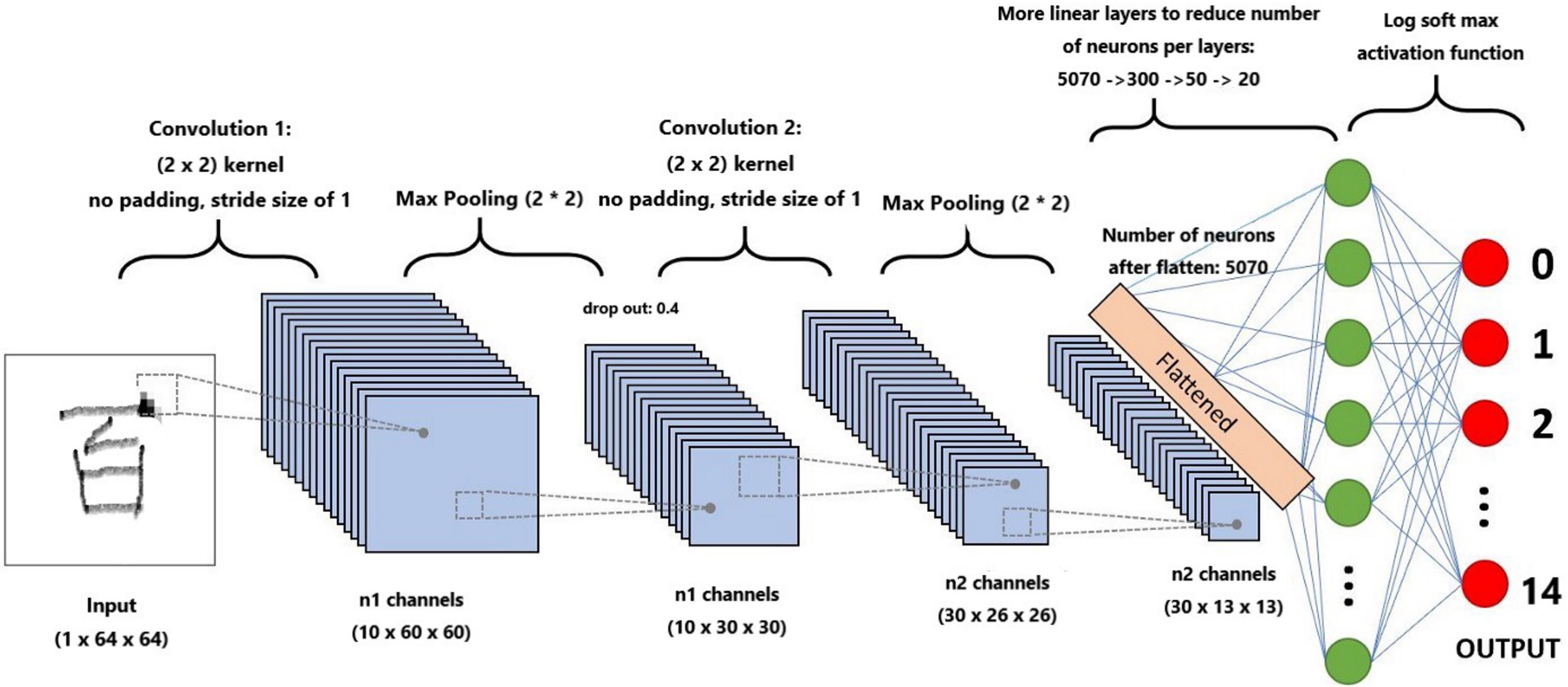
- Figure shown all the “zero” image
- Different position
- Different font size
- Different handwriting style..



CNN with PyTorch

Finally, we used Pytorch **Convoluted Neural Network** model and obtained a test accuracy of 89%.

```
CNN((dropout): Dropout(p=0.4, inplace=False)
 (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 (conv1): Conv2d(1, 5, kernel_size=(5, 5), stride=(1, 1))
 (conv2): Conv2d(5, 30, kernel_size=(5, 5), stride=(1, 1))
 (fc1): Linear(in_features=5070, out_features=300, bias=True)
 (fc2): Linear(in_features=300, out_features=50, bias=True)
 (fc3): Linear(in_features=50, out_features=20, bias=True)
 (activation): LogSoftmax(dim=1))
```


More linear layers to reduce number of neurons per layers: 5070 ->300 ->50 -> 20
Log soft max activation function
Convolution 1: (2 x 2) kernel no padding, stride size of 1
Max Pooling (2 * 2)
Convolution 2: (2 x 2) kernel no padding, stride size of 1
Max Pooling (2 * 2)
Number of neurons after flatten: 5070
drop out: 0.4
Flattened
0
1
2
14
Input (1 x 64 x 64)
n1 channels (10 x 60 x 60)
n1 channels (10 x 30 x 30)
n2 channels (30 x 26 x 26)
n2 channels (30 x 13 x 13)
OUTPUT

Model design plan. The numbers are not final or guarenteed to be accurate, and they are not perfectly presented by the figures
Picture adapted from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Result Present & Take-away

- Traditional models are outperformed by boosting, bagging methods
- Confirm that CNN achieve the best performance for image classification

Future improvement:

- Update CNN model with better structure and parameters (change padding, pooling… etc)
- Use a smaller sample data so it doesn’t take too long to train
- Try recurrent neural network, because of the natural writing order.

Algorithm(Model)	**Test Accuracy**
Binary: LogReg	~20%
Binary: Bagging & Voting	~37.48%, 52.81%
MLP	~32%
Baseline (LogReg)	36.2%
SVM	43.7%
Random Forest	61.8%
ADABoost	57.3%
XGBoost	62.7%
LightGBM	71.8%
CNN	89.0%

Links

Written Report: https://docs.google.com/document/d/1ouunkNe6c57gTHiGRekZm0cppU8DFarIOkeBktWPe3k/edit?usp=sharing

Github Repo:

https://github.com/COGS118B-character-classification/Chinese-MNIST