COMP5318 Machine Learning and Data Mining

Assignment 2 Group 50 Report

Tutors: Yixuan Zhang, Huilan Zhu

Group members: Qi Lu (490196302), Guo Shiming(500672949)

Group participation:

Qi Lu: finish data preprocessing for CNN, the ResNet50 model implementation and turning. Report writing.

Contribution percentage:50%

Guo Shiming: finish data preprocessing for SVM and Random forest, SVM and Random forest model implementation and turning. Report writing.

Contribution percentage:50%

Abstract

The report describes the analysis and implementation of image multi-classification tasks. Three different classifiers are designed, analysed and evaluated from scratch. After importing the data, different classifiers use different preprocessing methods. Since the labels of the dataset are unbalanced, the report also discusses the influence of classifiers between the original data and the resample data. In addition, this report discusses the advantages and disadvantages of each classifier, the selection of hyperparameters, the impact of changing hyperparameters of a particular classifier, and their results. Different parameters, such as recall, precision and F1 scores, are used to evaluate the best classifier.

Introduction

Dataset Overview

The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19. It was downloaded from the Kaggle website located at the URL https://www.kaggle.com/datasets/crawford/emnist. There are many datasets in the EMNIST dataset. The file "emnist-byclass-train" is chosen as the training dataset. It has 697932 rows and 785 columns. The file "emnist-byclass-test" is chosen as the test dataset. It has 116323 rows and 785 columns. The feature is from the second column to the 785th column in both files. One row of the feature means 28*28 pictures. The label is in the first column. The dataset has 62 labels, containing 26 upper and lower case letters and 10 numbers from 0 to 9.

These two datasets have the same image information, but are different in the number of images on each label. Both datasets have an uneven number of images per class, and there are more numbers than letters. The number of letters roughly equates to the frequency of use in daily life. Moreover, we can get a general understanding of the dataset through the following picture(Fig.1):

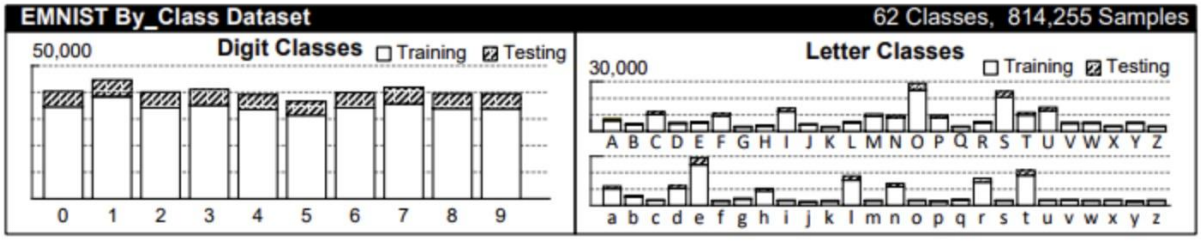


Fig.1. Labels distribution in training and testing data

In our assignment, we only choose 308016 rows of the training dataset and 70000 rows of the test dataset. There are two reasons for this. One is that we do not have enough memory to compute the whole dataset. The other is that the original dataset is imbalanced. We regenerate a 308016 rows samples from the original 90,000 rows of samples using the SMOTE algorithm, then we use the original 308016 rows of samples to fit our three other classifiers, and for the convolutional neural network, we compare the prediction performance between

the original imbalanced 308016 rows of data and the 308016 rows of balanced samples. Because a smaller dataset is used, this allows us to reduce the computation time.

relevance in diverse applications

Handwriting recognition has a numerous range of applications. It can help us read postal addresses, bank check amounts and forms in daily life. It will give us a big convenience. Many years ago, digital storage technology was not yet mature. People have to reserve the data on paper. Now people can transfer paper files to digital storage using handwriting recognition technology. Otherwise, we need many people to print on the computer line by line. Handwriting recognition technology can also have an application in exams. Students can write results on paper and transform them into the teachers' computers. Handwriting recognition influences every aspect of life. It can make our lives easy and convenient.

Method and result

We preprocess three datasets. Firstly, we use PCA, normalisation and standardisation to preprocess the dataset. Support Vector Machines and random forest are chosen to classify the preprocessed dataset. Secondly, We resize the dataset and normalise it. Resnet50 is chosen to classify the preprocessed dataset. Thirdly, we use SMOTE to resample it because the dataset is unbalanced. Resnet50 is chosen to classify the preprocessed dataset. Finally, we compare the results from different classifiers.

Previous Work

The EMNIST-byclass dataset used in this article is a character recognition dataset. This article uses support vector machine, random forest and convolutional neural networks as classifiers to solve this classification problem. Jie Zhang and Xiaohong Wu published k nearest neighbours SVM (SVM-KNN) methods.(1) It is a typical method to reduce the number of training procedures in SVM. First, the k nearest neighbours (KNN) of a query sample are found by calculating the distances of the query to all the training samples. Then to the k nearest neighbours, if the labels of these samples are different, the pairwise distance matrix between the k neighbours is computed and converted into a kernel matrix, and then directed acyclic graph SVM (DAGSVM) is trained by the kernel matrix. Compared with our SVM processing, we first use normalisation and standardisation and then let PCA equal 0.9 to reduce the dimension. SVM classifiers use this preprocessing data to classify.

In the research on handwritten character recognition on EMNIST-byclass datasets, a different approach using WAVEMIX is performed to achieve a better result than the current existing Convolutional neural network.(2) They use multi-scale 2D discrete wavelet transform (DWT) for spatial token mixing and introduce another inductive bias to improve the generalisation. This approach can generalise competitive results yet use fewer computational resources than convolutional neural network. The main difference between the WAVEMIX and the ResNet50 architecture is that WAVEMIX can achieve generalisation in fewer layers of wavelet transforms. The 2D DWT is a

linear transform with no learnable parameters that can extract features compared to the ResNet50, and it can still achieve the same level of model performance. Also, our ResNet50 model has 48 convolution layers, one max pooling layer and one average pooling layer; it needs all of these layers with large numbers of trainable parameters to achieve a good generalisation.

Methodology

Data Pre-processing techniques

Normalisation and standardisation

Normalisation means limiting the range of our data values to between [0,1], and we usually use the following formula(Fig.2). Standardisation means making the processed data conform to a standard normal distribution(Fig.3).

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Fig.2. Normalisation

$$x' = \frac{x - \mu}{\delta}$$

Fig.3. Standardisation

Normalisation and standardisation aim to reduce the magnitude of impact between different features. After using them, the next step of PCA will have a good performance in reducing dimensions.

Principal components analysis

PCA is the most common dimensionality reduction method. The main idea of PCA is to map n-dimensional features to k-dimensions(n>k). We constantly find new axes chosen in the direction of the most significant variance from the previous data. This can confirm that we constantly get the dimension that reserves the most content.

We usually choose PCA equals to 0.9 or 0.95. It makes the balance between the calculating time and the content. As the dataset has too much data, this will take much computational time when PCA equals 0.95. So we decided to use a PCA equal to 0.9 as our criterion. As you can see in the picture below(Fig.4), when the component equals 100, the explained variance is up to 0.9. At last, we reserved 100 components of our train dataset.

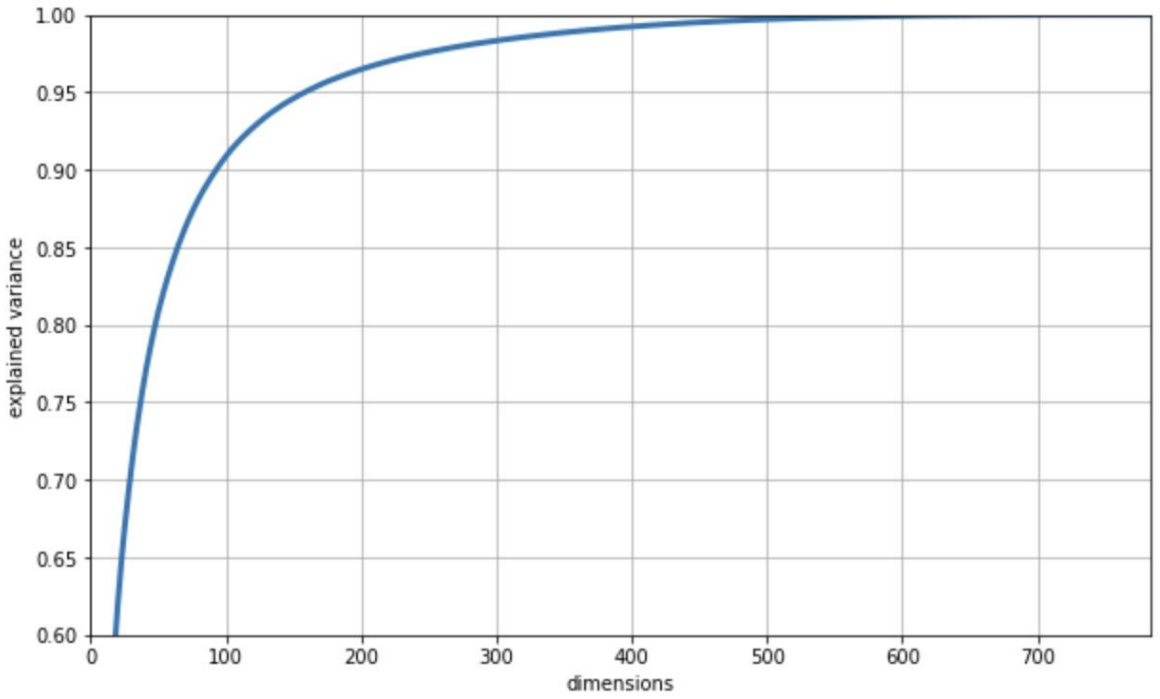


Fig.4. PCA

SMOTE

The dataset EMNIST-byclass has 62 classes of different characters(Fig.5), and the train data contains unbalanced classes, which means there exists a biassed or skewed in our training data. This unbalanced training might cause a problem when using classification models since most of the classification assumes the data to be equal in each class. Therefore a trick to balance the data is needed.

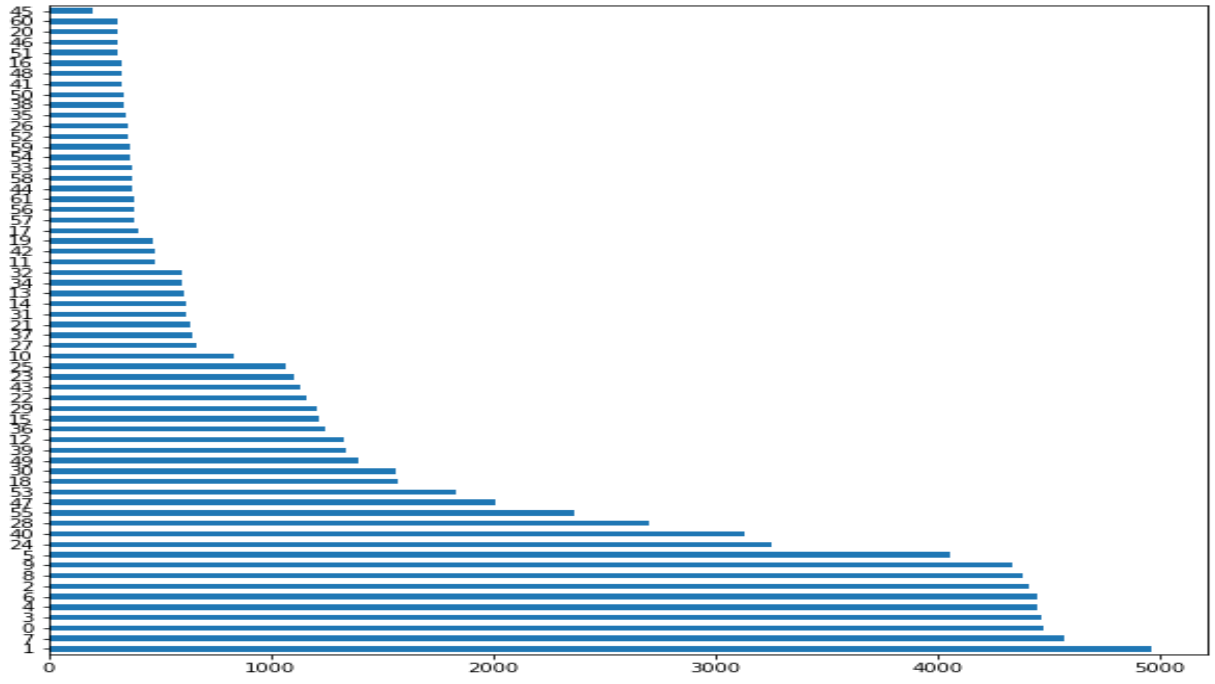


Fig.5. Labels distribution in training data

Our research conduct SMOTE oversampling to overcome the unbalanced training data. The SMOTE oversampling technique uses synthetic samples generated for the classes with fewer samples. It helps to overcome overfitting by random oversampling. It focuses on the feature space to generate new instances that would help interpolate between closed positive instances. The image below shows the new resample data set(Fig.6):

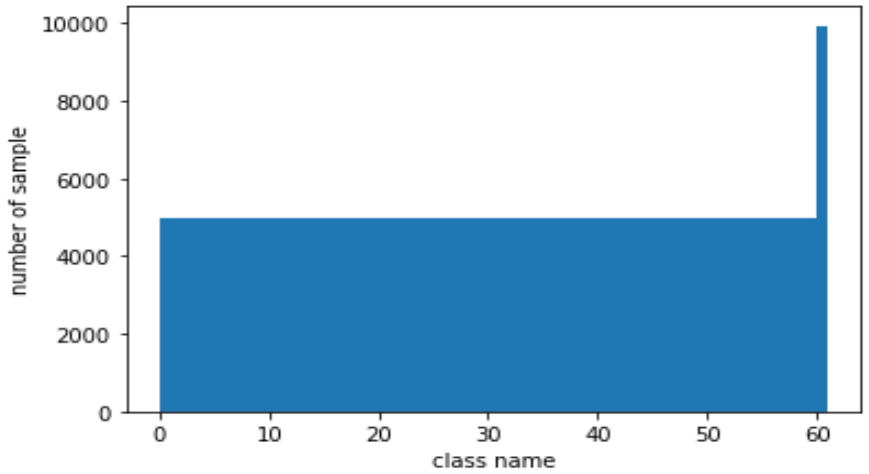


Fig.6. Labels distribution in resample data

Dimension expansion and Image resize.

In order to make sure the dataset of the EMNIST-byclass can be used in the ResNet50 Model, we need to change the dimension of the original dataset from 1 channel picture to 3 channel picture since the ResNet50 architecture takes in RGB image data. The numpy expand_dims function achieves this.

Also, the ResNet50 architecture requires a minimum image size of 32*32. Otherwise, the output image will be less than size 0 during the 5 stages of downsampling. This process is achieved by transferring the original array data to a tensorflow's tensor image and then using the tensor flow image. Resize function to resize each character image to a size 32*32 image.

Classification algorithm

Support Vector Machines

The basic idea of SVM learning is to solve for the separating hyperplane that correctly partitions the training dataset and has the most significant geometric separation. The figure(Fig.7) below shows that $w \cdot x+b = 0$ is the separation hyperplane. For linearly divisible datasets, there are many such hyperplanes (i.e. perceptrons), but the geometrically spaced separation hyperplane is the only one. When the new points become available later, the separating hyperplane will also make a reasonable classification.

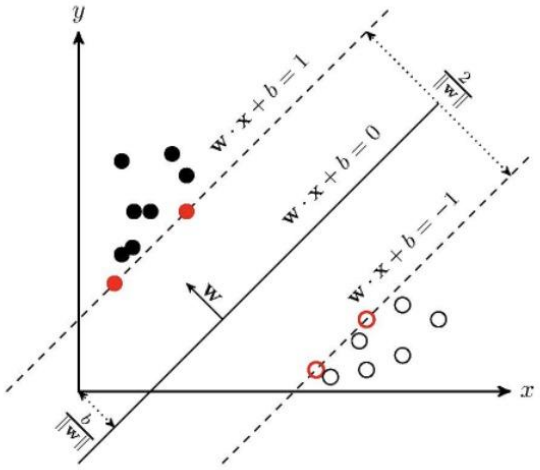


Fig.7. SVM principal

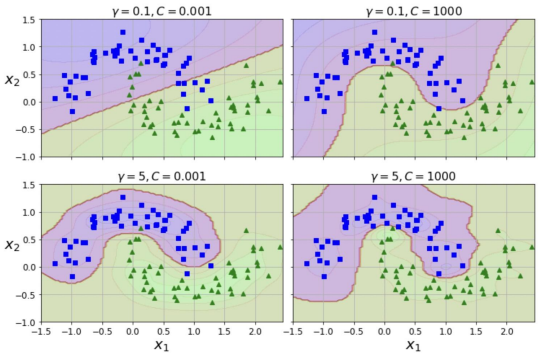


Fig.8. Gamma and C in SVM

The essential hyperparameters are 'C' and 'gamma' in SVM(Fig.8). The parameter 'C' controls the regularisation, and the parameter gamma controls the width of the Gaussian kernel. A small C means a very restricted model, where each point has a minor influence. This means that more points in the training set are not correctly predicted by the model, which may cause underfitting. A big C means a less restrictive model and a more significant influence on all points. If we choose a C big enough, all the points in the training dataset will predict right, but it may have a lousy performance in test data. A small gamma means a large radius of the bell. The instances have a large receptive field and influence. A small gamma can let up high accuracy in the training dataset but may cause overfit. On the contrary, a big gamma means a narrower bell shape. Each instance has a smaller receptive field and influence. A big gamma does not fit the train data very well, but may get good accuracy in test data.

Random Forest

Random Forest (Fig.9) is an algorithm that integrates multiple trees through the idea of ensemble learning. Its basic unit is the decision tree. A decision tree is a tree structure in which each internal node represents a judgement on an attribute, each branch represents the output of a judgement, and finally, each leaf node represents a classification result. The logic of the decision tree model is that starting from the root node, each feature of the instance is judged. According to the judgement result, the instance is assigned to its child nodes, at which point each node corresponds to a value of that feature, and so on recursively until the instance is assigned to a leaf node.

Each decision tree can be seen as a classifier. Then after inputting a sample in a random forest, every decision tree in the random forest will get its prediction result. The random forest selects the result with the highest ratio as the final output. Random forests build multiple decision trees and merge them to obtain more accurate and stable predictions. The most crucial hyperparameter in the random

forest is the number of decision tree. Random forest usually converges to lower generalisation errors as the number of base classifiers increase. If we continue to increase the number, it will only take more time in the training model. It is essential to select a suitable number of decision trees in random forest.

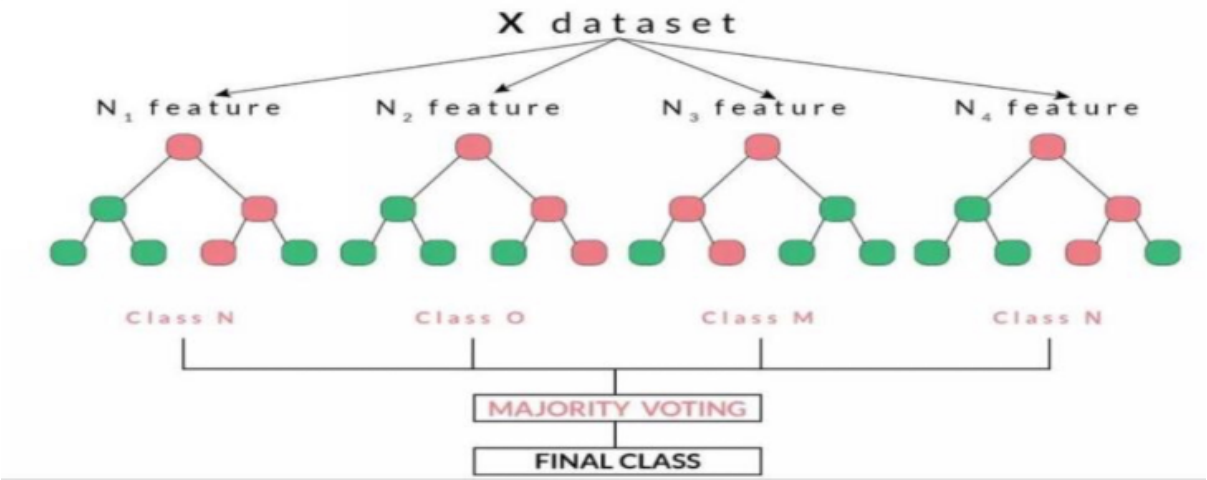


Fig.9. Random forest principal

Resnet 50

Resnet stands for Residual Network, a family of networks widely used in areas such as target classification and as part of the classical neural network backbone of computer vision tasks, of which the typical network is resnet50(Fig.10). Resnet50 has four groups of blocks, each with 3, 4, 6 and 3 blocks, each with three convolutional layers. In addition, the network starts with one convolutional layer and ends with a fully connected layer, so (3+4+6+3)* 3 + 1+1 = 50. This is also the origin of the 50 in resnet50.

In the picture, we show the construction of the resnet 50. On the left side, we show the overall structure. On the middle side, we show the structure of each stage. On the right side, we show the specific structure of the Bottleneck.

In stage 0, CONV means the convolutional layer. BN means the Batch Normalisation layer. RELU means activation function. MAXPOOL means the max pool layer. We first input the data which channel is 3, height and width are 224. After processing by conventional layer, Batch Normalisation layer, RELU activation function and max pool layer. We get the output which channel is 64, height and width are 56. In stage 1, there are 3 two types of Bottleneck structures. We first input the data shaped as channel 64, length and width 56, and put them into Bottleneck 1. In Bottleneck1(BTNK1 on the top right of the picture), we can get the first output through the three convolutional layers, batch normalisation layers, and RELU activation function on the left. We can get the second output through one convolutional layer and the batch normalisation layer on the right. Then we input these two outputs in the RELU activation function. Finally, we get the result shaped as channel64, length and width 56. After Bottleneck1, we input the result above to Bottleneck2(BTNK2 on the downright of the picture). In Bottleneck2, we put part of

the data into the three convolutional layers, batch normalisation layers, and RELU activation function on the left, then put this output and the other data into RELU activation function on the right. Finally, we can get the result. Moreover, we input this result to the Bottleneck2 again. After that, we can get the output of stage 1. In the same way, We pass the data through stage 2, 3, 4 and get the final output result.

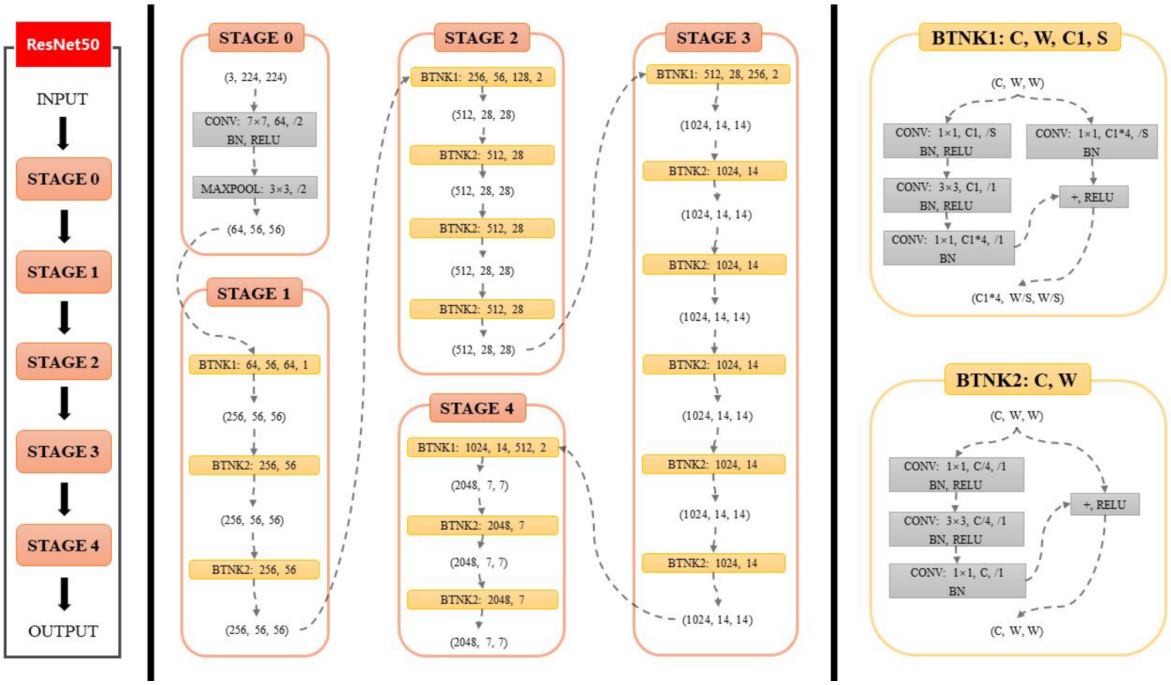


Fig.10. Resnet50 principal

Experiment

hardware and software specification:

In this experiment we are using the google colab pro (Fig.11)runtime, thus the system information is as follow:

CPU: Intel(R) Xeon(R) CPU @ 2.20GHz

RAM: 26692024 kB

GPU: Tesla P100-PCIE-16GB

The runtime type is with GPU hardware acceleration and runtime shape is high-RAM.

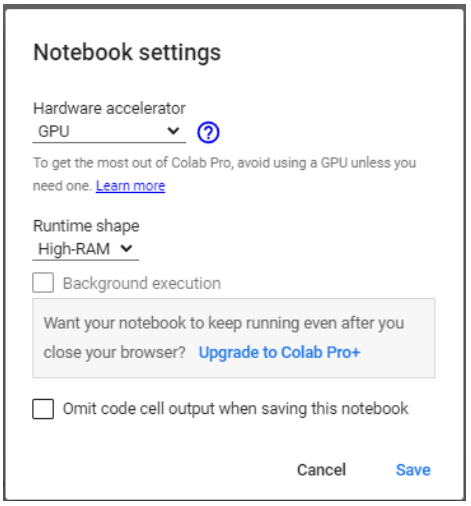


Fig.11. Notebook settings

SVM

After the 2 fold cross-validation, we find the best result is that 'C' equals 3 and the gamma equals 1. The score is 0.837.(Fig.11) We fit the SVM model with the best

hyperparameter. The accuracy of the test dataset is 0.37684. The F1 score of the test dataset is 0.37.

```
[CV 1/2] END ......................C=1, gamma=1;, score=0.827 total time=65.1min
[CV 2/2] END ......................C=1, gamma=1;, score=0.828 total time=65.1min
[CV 1/2] END ......................C=1, gamma=3;, score=0.829 total time=98.6min
[CV 2/2] END ......................C=1, gamma=3;, score=0.830 total time=97.9min
[CV 1/2] END ......................C=3, gamma=1;, score=0.836 total time=61.1min
[CV 2/2] END ......................C=3, gamma=1;, score=0.837 total time=62.2min
[CV 1/2] END ......................C=3, gamma=3;, score=0.828 total time=89.3min
[CV 2/2] END ......................C=3, gamma=3;, score=0.830 total time=87.6min
```

Fig.11. Time and score in each cv and hyperparameter

Random Forest

After the 2 fold cross-validation, we find the best result is '`n_estimators`

' equals 300. The score is 0.758. We fit the random forest model with the best hyperparameter. The accuracy of the test dataset is `0.46118. The F1 score` of the test dataset is `0.42`.

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
[CV 1/2] END ..................n_estimators=100;, score=0.749 total time= 1.3min
[CV 2/2] END ..................n_estimators=100;, score=0.750 total time= 1.3min
[CV 1/2] END ..................n_estimators=200;, score=0.755 total time= 2.6min
[CV 2/2] END ..................n_estimators=200;, score=0.755 total time= 2.6min
[CV 1/2] END ..................n_estimators=300;, score=0.757 total time= 3.8min
[CV 2/2] END ..................n_estimators=300;, score=0.758 total time= 3.8min
Best parameters: {'n_estimators': 300}
Best cross-validation score: 0.7573
Best estimator:
RandomForestClassifier(n_estimators=300, n_jobs=-1)
CPU times: user 1h 9min 51s, sys: 47.1 s, total: 1h 10min 38s
Wall time: 23min 41s
```

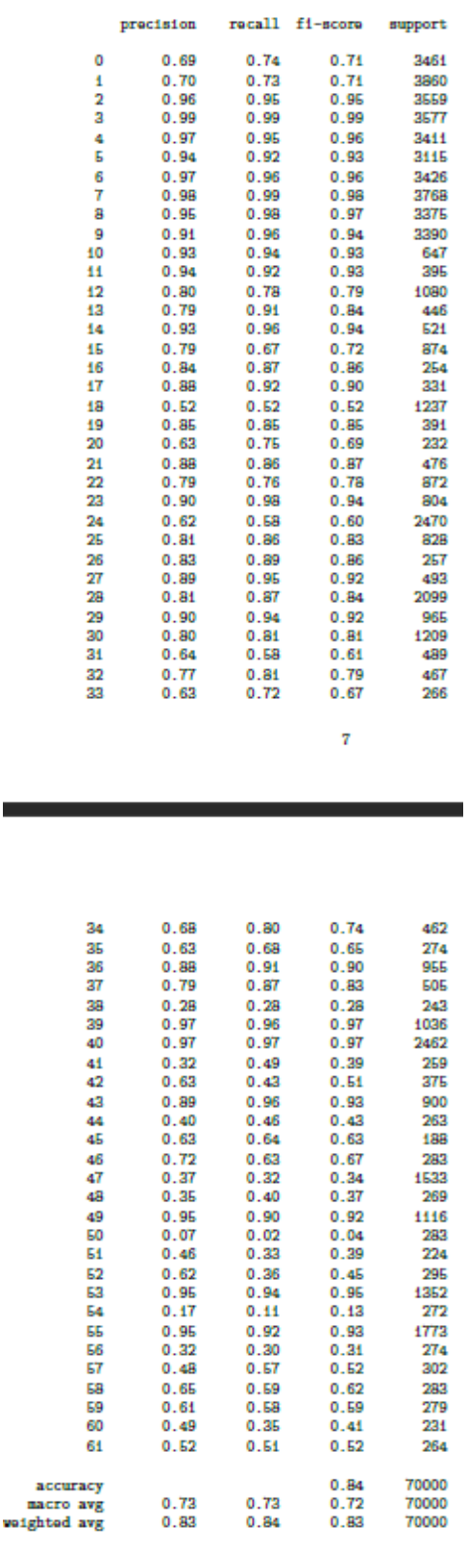
Resnet 50 (balanced and imbalanced)

In this experiment, the Resnet50 model was implemented through keras ResNet50 model, with no pre-train weight by setting the weights argument to None and not setting the parameter in the model to untrainable, allowing us to train the Resnet50 model on our own dataset.

After the convolution neural network, few layers were added to feed the data into a multilayer perceptron in order to help classify the image.

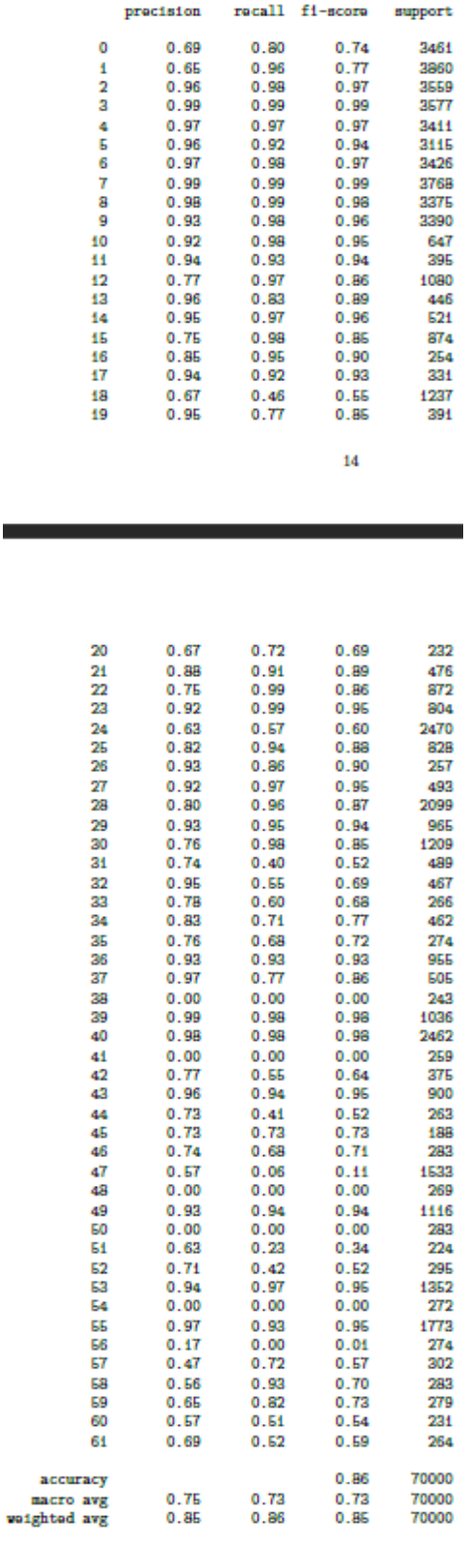
The hyperparameter tuning of the Resnet50 model is done by handwritten the turning function: **model_turning(lr, bs, base_model)** this will turn the model with different learning rate in Adam optimizers and different batch sizes to see which brings the best accuracy in the test data.

Our model informs us the best parameter of learning rate and batch size is 0.001 and 128. Thus the ResNet50 model we use on both balanced dataset and imbalanced dataset sets 0.001 as learning rate and 128 as batch size.



	precision	recall	f1-score	support
0	0.69	0.74	0.71	3461
1	0.70	0.73	0.71	3860
2	0.96	0.95	0.95	3559
3	0.99	0.99	0.99	3577
4	0.97	0.95	0.96	3411
5	0.94	0.92	0.93	3115
6	0.97	0.96	0.96	3426
7	0.98	0.99	0.98	3768
8	0.95	0.98	0.97	3375
9	0.91	0.96	0.94	3390
10	0.93	0.94	0.93	647
11	0.94	0.92	0.93	395
12	0.80	0.78	0.79	1080
13	0.79	0.91	0.84	446
14	0.93	0.96	0.94	521
15	0.79	0.67	0.72	874
16	0.84	0.87	0.86	254
17	0.88	0.92	0.90	331
18	0.52	0.52	0.52	1237
19	0.85	0.85	0.85	391
20	0.63	0.75	0.69	232
21	0.88	0.86	0.87	476
22	0.79	0.76	0.78	872
23	0.90	0.98	0.94	804
24	0.62	0.58	0.60	2470
25	0.81	0.86	0.83	828
26	0.83	0.89	0.86	257
27	0.89	0.95	0.92	493
28	0.81	0.87	0.84	2099
29	0.90	0.94	0.92	965
30	0.80	0.81	0.81	1209
31	0.64	0.58	0.61	489
32	0.77	0.81	0.79	467
33	0.63	0.72	0.67	266
34	0.68	0.80	0.74	462
35	0.63	0.68	0.65	274
36	0.88	0.91	0.90	955
37	0.79	0.87	0.83	505
38	0.28	0.28	0.28	243
39	0.97	0.96	0.97	1036
40	0.97	0.97	0.97	2462
41	0.32	0.49	0.39	259
42	0.63	0.43	0.51	375
43	0.89	0.96	0.93	900
44	0.40	0.46	0.43	263
45	0.63	0.64	0.63	188
46	0.72	0.63	0.67	283
47	0.37	0.32	0.34	1533
48	0.35	0.40	0.37	269
49	0.95	0.90	0.92	1116
50	0.07	0.02	0.04	283
51	0.46	0.33	0.39	224
52	0.62	0.36	0.45	295
53	0.95	0.94	0.95	1352
54	0.17	0.11	0.13	272
55	0.95	0.92	0.93	1773
56	0.32	0.30	0.31	274
57	0.48	0.57	0.52	302
58	0.65	0.59	0.62	283
59	0.61	0.58	0.59	279
60	0.49	0.35	0.41	231
61	0.52	0.51	0.52	264
accuracy			0.84	70000
macro avg	0.73	0.73	0.72	70000
weighted avg	0.83	0.84	0.83	70000

figure 12



	precision	recall	f1-score	support
0	0.69	0.80	0.74	3461
1	0.65	0.96	0.77	3860
2	0.96	0.98	0.97	3559
3	0.99	0.99	0.99	3577
4	0.97	0.97	0.97	3411
5	0.96	0.92	0.94	3115
6	0.97	0.98	0.97	3426
7	0.99	0.99	0.99	3768
8	0.98	0.99	0.98	3375
9	0.93	0.98	0.96	3390
10	0.92	0.98	0.95	647
11	0.94	0.93	0.94	395
12	0.77	0.97	0.86	1080
13	0.96	0.83	0.89	446
14	0.95	0.97	0.96	521
15	0.75	0.98	0.85	874
16	0.85	0.95	0.90	254
17	0.94	0.92	0.93	331
18	0.67	0.46	0.55	1237
19	0.95	0.77	0.85	391
20	0.67	0.72	0.69	232
21	0.88	0.91	0.89	476
22	0.75	0.99	0.86	872
23	0.92	0.99	0.95	804
24	0.63	0.57	0.60	2470
25	0.82	0.94	0.88	828
26	0.93	0.86	0.90	257
27	0.92	0.97	0.95	493
28	0.80	0.96	0.87	2099
29	0.93	0.95	0.94	965
30	0.76	0.98	0.85	1209
31	0.74	0.40	0.52	489
32	0.95	0.55	0.69	467
33	0.78	0.60	0.68	266
34	0.83	0.71	0.77	462
35	0.76	0.68	0.72	274
36	0.93	0.93	0.93	955
37	0.97	0.77	0.86	505
38	0.00	0.00	0.00	243
39	0.99	0.98	0.98	1036
40	0.98	0.98	0.98	2462
41	0.00	0.00	0.00	259
42	0.77	0.55	0.64	375
43	0.96	0.94	0.95	900
44	0.73	0.41	0.52	263
45	0.73	0.73	0.73	188
46	0.74	0.68	0.71	283
47	0.57	0.06	0.11	1533
48	0.00	0.00	0.00	269
49	0.93	0.94	0.94	1116
50	0.00	0.00	0.00	283
51	0.63	0.23	0.34	224
52	0.71	0.42	0.52	295
53	0.94	0.97	0.95	1352
54	0.00	0.00	0.00	272
55	0.97	0.93	0.95	1773
56	0.17	0.00	0.01	274
57	0.47	0.72	0.57	302
58	0.56	0.93	0.70	283
59	0.65	0.82	0.73	279
60	0.57	0.51	0.54	231
61	0.69	0.52	0.59	264
accuracy			0.86	70000
macro avg	0.75	0.73	0.73	70000
weighted avg	0.85	0.86	0.85	70000

figure 13

Since the original dataset is imbalanced we decide to compare if converting the dataset to balanced will bring improvement on predicting classes in test data.

An oversampling technique was performed on 90000 samples from the original train dataset and the result of 308016 samples which is consistent with the sample size of the other model we used. The Resnet50 model on a balanced dataset gives us an accuracy of 0.84 and the f1-score of each class(figure 12). We found that the f1-score on each class is relatively high with few exceptions on class 38, 41, 47, 50, 54, 56. Which indicates to us that the model works fine under the balanced dataset and gives us a relatively good result with an overall accuracy of 0.84. Since the dataset is balanced thus we suggest that if we want to achieve better results on this model, we need to have better feature extraction on those classes with low f1-score to better differentiate them from other classes that they have been miss classified as.

The ResNet50 model has also been performed on the imbalanced data, the classification report is shown in (figure 13). This model is performed on the subsampling 308016 samples on the original dataset inorder. Thus the dataset is imbalanced.

As the result shows, this model consists of more classes that have extremely low f1-score compared to the balanced dataset, this is a result of undersampling on some classes since some classes have only a couple hundreds of data to train the model compared to some have a couple thousands samples to train. Therefore the weight on these classes with small amounts of samples are very low and result in the low f1-score in these classes. However we observed that the overall accuracy is higher than the model performs on the balanced dataset. After discussion and investigation on the test dataset, we believed that it is because the test data also have the same distribution as the original train datasets and the classes weighted higher in this model have more test samples to predict and thus resulted with a higher overall accuracy.

The runtime for the Resnet50 on the balanced dataset took average 100s per epoch and 29s on predicting the testset with 70000 samples under the batchsize 128.

The runtime for the Resnet50 on the imbalance dataset took average 99s per epoch and 28s on predicting the same testset with 70000 samples under the batchsize 128.

Thus the ResNet50 model performs similarly on both balanced and imbalanced datasets.

comparison

classifier	accuracy	F1 score	rank
SVM	0.38	0.37	4
Random forest	0.46	0.42	3
resnet50 with original data	0.86	0.86	1
resnet50 with resample data	0.84	0.83	2

Svm

As we can see in the table, SVM has the worst performance. The support vector machine algorithm is a two-class classification algorithm, but in our dataset, we have 62 labels. We used a combination of multiple two-class support vector machines to train the model and predict the labels. This may contribute to why SVM has the worst performance of all models we used with a 38% accuracy.

RandomForest

For data with different attributes that take more value divisions will have a greater impact on the random forest, so the attribute weights produced by the random forest on such data are not credible. This may contribute to why random forests do not have good performance only predicting correctly on a 46% accuracy.

ResNet50

In our dataset, ResNet50 models have the best performance. ResNet50 has three advantages over traditional machine learning methods: ResNet50 can scale data efficiently, it does not require feature engineering, it is adaptable and easy to transform. So deep neural networks can achieve accuracy that far exceeds those traditional machine learning methods. And the ResNet50 performed better on the imbalance dataset. With the top accuracy of 86%, exceeds the ResNet50 on the balanced dataset that only has a 84% accuracy.

reflect on our design choices

1. Labels of the original dataset are imbalanced. We get the resample dataset by using the SMOTE algorithm, but when we use the resample dataset in CNN, we find the prediction performance is lower than the original dataset. If we choose a suitable resample algorithm, we may get a higher prediction performance in CNN
2. We only choose part of the dataset to fit and predict in all classifiers. In svm and random forest, we use a 2 fold cross-validation. Because if we choose an overall dataset, we can not fit the model by our hardware and software environment. We may choose a suitable classifier, which needs a lower RAM and calculation time. Thus we can use the entire original dataset.
3. We find the F1 scores for certain labels are extremely low. It is because when we use ResNet50, The algorithm extracts features of certain labels which are similar to features of some different labels that the model could not separate them clearly. This contributes to low F1 scores in some labels. We may choose a better classifier to increase the prediction performance or we could use pre-processing techniques to make the samples of handwriting look more different to those wrongly classified classes.

Conclusion

The CNN model has way better accuracy than traditional machine learning methods and ensemble methods.

We believe the ResNet50 model performed on a balanced dataset reflects the most normal behaviour, since if we use these two CNN models to predict the handwritten characters, we should not expect the input of the data to have the same distribution as our original training data. Therefore although the ResNet50 trained on imbalanced datasets has a higher accuracy on these test datasets, we believe in real life this accuracy would not hold and the ResNet50 trained on a balanced dataset will have a better prediction accuracy.

Future work :

We may focus on how to better differentiate the classes with a better feature extraction strategy. And find a way to fit more data into the CNN with limited hardware and software resources. We may also focus on a good way to preprocess the dataset, for example, we can use a more accurate resample method.

Reference

1. Zhang J, Wu X, Yu Y, Luo D. A Method of Neighbor Classes Based SVM Classification for Optical Printed Chinese Character Recognition. [cited 2013 May 17] PLoS ONE 8(3): e57928. doi:10.1371/journal.pone.0057928 Available from: https://sydney.primo.exlibrisgroup.com/discovery/fulldisplay?docid=cdi_scopus_primary_372537458&context=PC&vid=61USYD_INST:sydney&lang=en&search_scope=MyInst_and_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,character%20recognition,%20svm&facet=rtype,include,articles&offset=0

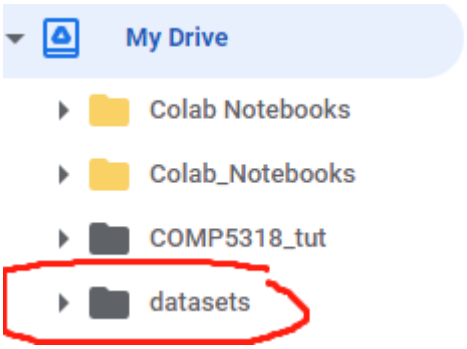
2. Pranav Jeevan P, Amit Sethi. WAVEMIX: RESOURCE-EFFICIENT TOKEN MIXING FOR IMAGES. arXiv:2203.03689v1 [Internet]. 2022 [cited 2022 May 7]. Available from: https://arxiv.org/pdf/2203.03689v1.pdf

Appendix:

Download file from: https://www.kaggle.com/datasets/crawford/emnist

There are two file needed:"**emnist-byclass-test.csv**" and "**emnist-byclass-train.csv**"

After downloading these two files, please upload them to the google drive where you need to create a directory named "datasets" and place these two files in the datasets directory As shown in below figure.



How to run the code files

For group50_best_algorithm1.ipynb file, You need to have google colab pro in order to run this, change the runtime type to GPU and use high-RAM runtime shape.

The first code chunk is to connect you to the google drive with datasets directory and two files inside. Import all necessary libraries using the second code chunk and you can check if you are connected to a gpu using the third code chunk and check if you have the file in the right directory using the fourth code chunk.

The rest of the model require you to run each code chunk once in order from top to bottom and if you do not want to spend time waiting the hyperparameter tuning you

can skip the code chunk followed the text **”Hyperparameter turning on learning rate of Adam optimizer and the batch size”**

Figure : you can skip this code chunk if you don’t want to wait for turning

The two models in this file are ordered by ResNet50 on balanced datasets followed by ResNet50 on imbalance datasets.

For group50_other_algorithms.ipynb file. The first code chunk is to bind colab with google cloud drive. Import all necessary libraries using the second code chunk. The third code chunk is to print file names in gdrive. We import the train and test dataset in the forth code chunk. The fifth code chunk shows the numbers of each label. Select features and labels from the train and test dataset using the sixth code chunk. Later we show the shape of each dataset in the seventh code chunk. Next part is for the preprocessing process, the aim of the eighth code chunk is normalisation and standardisation for the feature in train and test dataset. In the ninth code chunk, we show the relationship between explained variance and dimensions. In the tenth code chunk, we decide to choose a minimal 90% of the principal components. The next three code chunks are hyperparameter tuning. In the last two code chunks, we show the model with the best hyperparameter.

[LibSVM]Accuracy on test set y: 0.37684

	precision	recall	f1-score	support
0	0.51	0.61	0.56	3461
1	0.55	0.46	0.50	3860
2	0.17	0.25	0.20	3559
3	0.48	0.68	0.56	3577
4	0.30	0.35	0.32	3411
5	0.23	0.23	0.23	3115
6	0.74	0.77	0.75	3426
7	0.66	0.63	0.64	3768
8	0.39	0.56	0.46	3375
9	0.51	0.43	0.47	3390
10	0.26	0.44	0.33	647
11	0.19	0.17	0.18	395
12	0.30	0.28	0.29	1080
13	0.25	0.20	0.22	446
14	0.08	0.11	0.09	521
15	0.10	0.18	0.13	874
16	0.34	0.44	0.38	254
17	0.33	0.21	0.26	331
18	0.32	0.09	0.14	1237
19	0.20	0.25	0.22	391
20	0.03	0.03	0.03	232
21	0.07	0.03	0.04	476
22	0.59	0.75	0.66	872
23	0.19	0.26	0.22	804
24	0.50	0.35	0.41	2470
25	0.33	0.14	0.20	828

26	0.19	0.32	0.24	257
27	0.14	0.31	0.20	493
28	0.42	0.37	0.40	2099
29	0.37	0.26	0.31	965
30	0.20	0.12	0.15	1209
31	0.30	0.29	0.30	489
32	0.34	0.28	0.31	467
33	0.12	0.14	0.13	266
34	0.18	0.16	0.17	462
35	0.01	0.01	0.01	274
36	0.15	0.27	0.19	955
37	0.29	0.32	0.31	505
38	0.20	0.00	0.01	243
39	0.12	0.09	0.10	1036
40	0.56	0.48	0.52	2462
41	0.12	0.03	0.05	259
42	0.08	0.16	0.11	375
43	0.56	0.34	0.42	900
44	0.53	0.12	0.20	263
45	0.07	0.02	0.03	188
46	0.30	0.20	0.24	283
47	0.24	0.02	0.03	1533
48	0.50	0.01	0.02	269
49	0.38	0.69	0.49	1116
50	0.00	0.00	0.00	283
51	0.38	0.08	0.14	224
52	0.10	0.11	0.11	295
53	0.16	0.09	0.11	1352
54	0.00	0.00	0.00	272
55	0.41	0.36	0.39	1773
56	0.08	0.00	0.01	274
57	0.18	0.13	0.15	302
58	0.34	0.05	0.08	283
59	0.13	0.15	0.14	279
60	0.06	0.04	0.05	231
61	0.06	0.02	0.02	264
accuracy			0.38	70000
macro avg	0.27	0.24	0.24	70000
weighted avg	0.38	0.38	0.37	70000

Figure: scores of SVM

Random forest ensemble - accuracy on test set:
0.4611857142857143

	precision	recall	f1-score	support
0	0.53	0.75	0.62	3461
1	0.63	0.83	0.72	3860
2	0.21	0.42	0.28	3559
3	0.37	0.66	0.48	3577
4	0.40	0.71	0.51	3411
5	0.21	0.26	0.23	3115
6	0.62	0.92	0.74	3426
7	0.62	0.83	0.71	3768
8	0.33	0.48	0.39	3375
9	0.63	0.55	0.59	3390
10	0.44	0.34	0.38	647
11	0.50	0.00	0.01	395
12	0.43	0.20	0.27	1080
13	0.73	0.07	0.12	446
14	0.39	0.02	0.04	521
15	0.34	0.10	0.15	874
16	0.89	0.10	0.18	254
17	0.67	0.04	0.08	331
18	0.56	0.21	0.31	1237
19	0.67	0.07	0.13	391

20	0.64	0.11	0.18	232
21	0.60	0.05	0.09	476
22	0.61	0.81	0.69	872
23	0.37	0.28	0.32	804
24	0.47	0.33	0.39	2470
25	0.50	0.07	0.12	828
26	0.96	0.09	0.17	257
27	0.43	0.11	0.17	493
28	0.57	0.42	0.49	2099
29	0.51	0.37	0.43	965
30	0.35	0.16	0.22	1209
31	0.51	0.15	0.23	489
32	0.52	0.46	0.49	467
33	0.44	0.09	0.15	266
34	0.51	0.08	0.13	462
35	0.00	0.00	0.00	274
36	0.61	0.30	0.40	955
37	0.67	0.15	0.25	505
38	0.00	0.00	0.00	243
39	0.18	0.07	0.10	1036
40	0.61	0.68	0.64	2462
41	0.00	0.00	0.00	259
42	0.00	0.00	0.00	375
43	0.68	0.51	0.59	900
44	0.88	0.09	0.16	263
45	0.78	0.04	0.07	188
46	0.50	0.02	0.05	283
47	0.39	0.05	0.09	1533
48	0.00	0.00	0.00	269
49	0.53	0.77	0.63	1116
50	0.00	0.00	0.00	283
51	1.00	0.00	0.01	224
52	0.00	0.00	0.00	295
53	0.31	0.25	0.28	1352
54	0.00	0.00	0.00	272
55	0.57	0.60	0.59	1773
56	0.00	0.00	0.00	274
57	0.59	0.05	0.10	302
58	0.68	0.13	0.22	283
59	0.35	0.04	0.07	279
60	0.00	0.00	0.00	231
61	0.89	0.03	0.06	264
accuracy			0.46	70000
macro avg	0.46	0.24	0.25	70000
weighted avg	0.47	0.46	0.42	70000

Figure: scores of random forest