COMP5318 Machine Learning and Data Mining
Assignment 1
Group:34
Tutors: Yang Lin, Iwan Budiman

Shubin Du(480190510)   Jingjing Wu(480509446)
Liangchun Yu(480317999)

1 Abstract

This project is to classify a dataset of grey-scale images to different labels with supervised classifiers. We first pre-process the data with Normalisation and Principal Component Analysis(PCA), then classify our data with K-Nearest-Neighbors (KNN) and Logistic Regression using One-Versus-One (OVO) and One-Versus-All (OVA) methods.

2 Introduction

2.1 Aim of the study

With the ever increasing amounts of data becoming available, there is good reason to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress (Alex and S.V.N., 2008). In this project, our goal is to implement supervised classifiers to classify the categories of grey-scaled input data, with efficiency on accuracy and time spent. Our training dataset consists of 30,000 gray-scale images, each with the size of 28x28, and split into 10 different categories. The raw data is large in dimension, and the computing would still be time consuming, hence preprocessing the data would be more cost-effective. The training is based on the 30,000 gray-scale images(images_training.h5) and the 30,000 labels(labels_training.h5), where the validation is based on the testing data(images_testing.h5, labels_testing2000.h5).

2.2 Importance of the study:

Machine learning is used in numerous fields, making predictions through mathematical models with the benefit of growing computing speed of nowaday computers. Classification is an important field in machine learning, used in wide varieties of fields, such as identifying mails as "spam" or "non-spam", or classifying patients to different diagnosis based on age, syndromes, medical history, etc. Our project of classifying grey-scale images is within the scope of image classification, which is used in robotics, such as self-driving cars to recognize obstacles, traffic lights, or borders on the road. It is also widely used for face-recognition, for example security applications, or social-media websites such as Facebook.

3 Methods

3.1 Method Overview

We pre-process our data, and implement different classifiers and optimizers to get the best result.

3.2 Pre-processing:

Data preprocessing is an essential step in Machine Learning since the useful information and the quality of data that can be derived from it will directly affect the way our models learn and the performance of our models. Hence, it is important that we process our raw data before we feed it into our models.

3.3 Data Joining:

Our training data consists of two files(images_taining.h5, labels_training.h5), hence we combine the image data and labels data, so we can get a clear glimpse of the data and the labels it belongs to.

3.4 Data normalization/standardisation

As a integral step of machine learning, standardizing data can help transform the data to better "behave". There are two ways for rescaling and normalizing data:

One way is to scale all numeric variables to the range [0, 1] by this given formula:

$$X_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Another way to normalize the data is by subtracting the mean and dividing by the standard deviation just like we do to get from a Gaussian distribution to the standard normal distribution with mean = 0 and standard deviation =1:

$$X_{norm} = \frac{x - \mu}{\sigma}$$

We chose the first normalisation method and transformed it into a new normalized dataset.

3.5 PCA

PCA is a dimension reduction analysis that is used to reduce the dimensionality of large data sets to a smaller ones but still remaining most of the information. We reduce the numbers of less relevant variables in exchange of a faster computing speed but still maintaining as much information as possible, by linearly transforming an original data to a set of dimensionally independent data.

Step 1: Normalize the data

Since we have already normalized the dataset in advance, we can skip this step.

Step 2: Calculate the covariance matrix

Calculate the covariance matrix pairing each variables to see if some of the data are including redundant information with this formula:

$$cov\_mat = \frac{(X_{norm}).T.dot(X_{norm})}{m - 1}$$

Where m is the shape of the matrix, which is 784. Note that we have the variance of the initial variable on the main diagonal, and the upper and lower triangle of the matrix are symmetric.(Cov(i,i) = Var(i), Cov(i,j) = Cov(j,i) where i, j is the rows and columns of the matrix).

Step 3: Compute the eigenvectors and eigenvalues

We try to find the largest possible variance, which includes the most information. Our first principal component(PC1) is the line that maximizes variance, and the second component (PC2)is the line perpendicular to PC1 which is the second highest in variance. And so on we calculate *m* PCs. The eigenvector is the direction of axes of the PC line, and eigenvalue is the coefficient of the of the eigenvector. Then we divide each eigenvalue to the sum of all eigenvalues, getting the portion of each PC in the variance of the data. Eigenvectors can be generated through: *np.linalg.eig(cov_mat)*

Step 4: Form a feature vector

From the following step, we have our eigenvectors and eigenvalues in descending order, the feature vector consists of eigenvectors of the components we choose to keep, so we have a vector with a smaller dimension.

Step 5: Form dimension reduced data set

After generating the feature vector, we reorient the original data from the original axes to the ones of the principal components, done by the following equation:

$$final\_data = (Feature\_vec).T.\,dot\,(X_{norm}).T$$

3.6 Classifier

3.6.1 Overview of Logistic Regression:

First of all, in order to solve this multiclass classification problem, we implement logistic regression and using One-Versus-All (OVA) method which expand the binary Logistic Regression classifier. As we have a big training data and logistic regression model is efficient for the time and internal storage required. But the disadvantage is this method is put the other labels as one label except the positive one label we take out. This is easy to have biases because the binary classification see unbalanced distribution when the set of the negatives is massive and larger than the positive . Hence in order to increase the accuracy, we also try the One-Versus-One (OVO) method to make a contrast, which is just choose two of all labels to train for each iteration.

3.6.2 Hypothesis Representation

We treat the multiclass classification as the binary classification (choose one label as 0 and the others as 1). We train theta using the sigmoid function, which function is used to represent the probability of label 1.
hypothesis function is:

$$h_{\theta}(x) = g(\theta.T * x) \qquad \alpha = g(z) = 1/(1 + e ** (-z))$$

$$P(y = 1 | x; \theta) = \alpha$$

$$P(y = 0 | x; \theta) = 1 - \alpha$$

OVA method trains k regression model for the k different labels and sign each test instance with the bigger probability. For OVO method trains ½*k*(k-1) classifiers as there are kinds of composite(each choose two of them) for the k labels.

3.6.3 Cost Function

We use cross entropy as the cost function, which is fit for binary classification. Also as the derivative of this function including log could be much quicker for each iteration the formula is given below:

$$J(\theta) = -1/m * \sum[y log\ (h\theta(x)) + (1 - y) log(1 - h\theta(x))] \ \ for\ all\ \ x \in X, y \in Y$$

To avoid overfitting, we use l2 regularization which is easy to implement and as l2 regularization is good to treat the outlier which means useful to resolve high variance problem.

$$J(\theta) = -A/m \sum[y log\ (h\theta(x)) + (1 - y) log(1 - h\theta(x))] \ - \lambda/2 * ||\theta|| ** 2$$

$$for\ all\ x \in X, y \in Y$$

3.6.4 Optimisation Methods

We use gradient descent to be the optimisation method and deal with it as the baseline of our optimisation method, comparing with momentum and TNC methods.

3.6.5 Mini-batch Gradient Descent Formula (Baseline)

Instead of doing the normal gradient descent, we decided to use Mini Batch Gradient Descent. It is a variation of gradient descent algorithm, but it splits the training dataset into small datasets which we call them “mini batches” and use them to calculate error and update coefficients. In our implementation of Mini-batch Gradient Descent, we have mini batches of 100 each.

$$\theta := \theta - learningRate * \nabla J(\theta)$$
$$\nabla J(\theta) = X.T * (prediction - Y)/m$$

The default learningRate is 0.1 and number of iteration is 4000. When we implement L2 regularization(if overfitting), there is one more item added to the derivative of J(θ) is $\lambda * \theta$.

3.6.6 TNC

TNC is a non-linear, bound constrained optimizer. When one is using it, a function to minimize must be provided. This function must take one argument: a coordinate list to evaluate. Everytime it must return a tuple where the first element is the value of the function and the second element is the gradient as values. If it does not return a tuple like this, it abort the minimization.

3.6.7 Momentum

Momentum is a method that helps accelerate SGD and dampens oscillations. It adds a fraction γ of the update vector of last step to the current update vector. Using momentum is like a ball going down hill, the momentum term increase for dimensions whose directions are the same and reduce updates for dimensions whose directions are not the same (change direction). Therefore, we can get a faster convergence. The formula is given below:

$$v_t = \gamma * v_{t-1} + \alpha * \nabla_\theta J(\theta) \qquad \theta = \theta - v_t$$

3.6.8 K-Nearest-Neighbors(KNN)

K-Nearest-Neighbors(KNN) is an algorithm based on feature similarity, an object is classified as the most common class of its k nearest neighbors. KNN can be used in handwriting detection, image or video detection, etc.
First we calculate the euclidean distance of each neighbor:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

where n is the number of features, (p,q) is the value of the specific feature of the test sample and the neighbor. Then we calculate the K nearest neighbors, which is the K smallest distances from our test sample. We will optimize K to get the best accuracy.

4 Experiments and Results

4.1 Overview

The pre-processing of all the trainings are the same, we reshape the data to 784-dimension before doing the normalization. When we use PCA, we will keep 90% principal component, this will decrease the dimension of each instance of to 134. For all the experiments, we use the first 2000 examples as the validation data. For the logistic model, we find the performance of OVO is better than the OVA. The highest accuracy is the model KNN(with accuracy 84.78%). The following table gives the metrics of the three models, all with the best optimization and parameters.

	Accuray(%)	precision(%)	recall(%)	F1_score(%)	Time(s)
Baseline	73.85	74	76	73	75.3
OVO	78.45	78	78	78	97.3
KNN	84.78	84	91	84	182

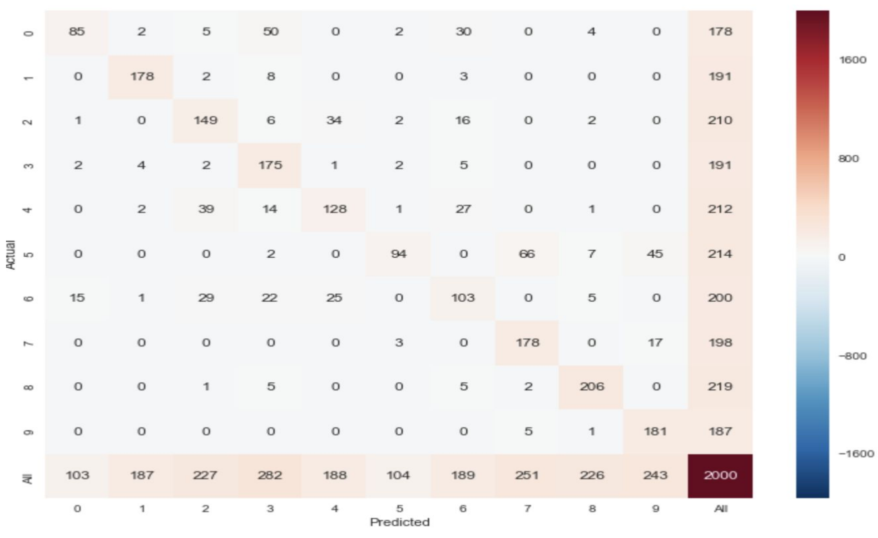
4.2 Baseline System

For baseline system, we use (One-versus-All) OVA method and gradient descent as its optimisation method. For every classifier, the learning rate is 0.1, and iterate 4000 times to get the minimum weight. Finally, we get the accuracy 73.85%, and have the precision, recall, F1-score with 74%, 76%, 73% separately. The total time for training and testing is 320.3s. The following table the gives values of Precision, Recall, F1-score and time.

Label	0	1	2	3	4	5	6	7	8	9
Precision(%)	48	93	71	82	50	44	52	90	94	97
Recall(%)	52	95	66	62	58	90	35	71	91	74
F1-score(%)	61	94	68	74	64	59	53	79	93	84
Time(s)	21.3	12.8	16.4	17.2	29.2	15.9	26.1	17.5	17.9	18.1

Table 1: Precision, Recall, F1-score and time spending for training classifier of each label.

The average time running for each label is all around 30s. This model shows better performance of labels 1, 3, 7, 8, 9, but does not perform good on labels 0, 2, 4, 5, 6. The recall of the label 6 is just 35% which means that label 6 is frequently predicted as other labels. The label 0, 4 and 5 have the same situation with label 6. The precision of label 7, 8, 9 are more than 90%, which illustrates that the model rarely predicted the other labels as them.



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	All
0	85	2	5	50	0	2	30	0	4	0	178
1	0	178	2	8	0	0	3	0	0	0	191
2	1	0	149	6	34	2	16	0	2	0	210
3	2	4	2	175	1	2	5	0	0	0	191
4	0	2	39	14	128	1	27	0	1	0	212
5	0	0	0	2	0	94	0	66	7	45	214
6	15	1	29	22	25	0	103	0	5	0	200
7	0	0	0	0	0	3	0	178	0	17	198
8	0	0	1	5	0	0	5	2	206	0	219
9	0	0	0	0	0	0	0	5	1	181	187
All	103	187	227	282	188	104	189	251	226	243	2000

Figure 1: OVA

The Figure1 shows the confusion matrix of the baseline. This picture could explain the above table more precisely. We can see that a lot of label 6 data points are predicted as labels 2, 3, 4, and some label 0 data points are classified as label 3 and 6. There are also a lot 5 being predicted as labels 7 and 9. Also there are some different labels being labeled as 4 and 5, most of them from 2, 6, 7, this decreases their precision value.

4.3 One-versus-One(OVO)

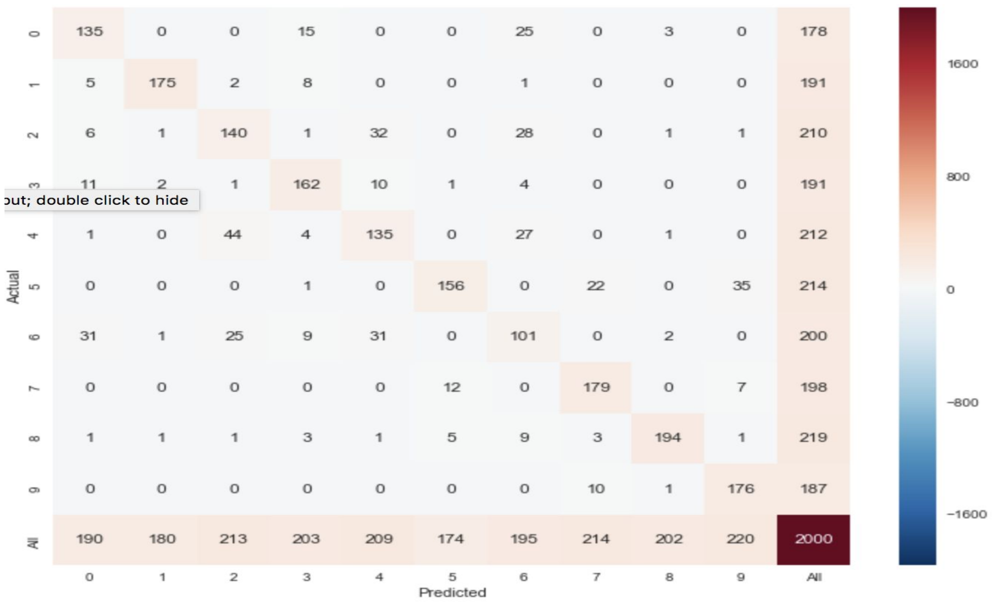
OVO algorithm take each possible binary pair of classes and build an ensemble. Therefore, we need to train in total $\frac{1}{2} \times k \times (k-1)$ classifiers for a k-multiclass problem, which is 45 here. During each label assigning stage, the classifier will do majority voting the two classes (which contrast to OVA since each classifier predicts probability in OVA classifier). Then, all these classifiers will be applied to an unseen sample and then the class that has been predicted the most will be the result.

The time spending for OVO strategy compared to OVA is increased from around 100s(use TNC optimisation). Also, the accuracy using gradient descent manually is 78.45%, which increased 5% compared to OVA. The average precision, recall and F1-score are all 78%.

Label	0	1	2	3	4	5	6	7	8	9
Precision(%)	76	92	67	85	64	73	50	90	89	94
Recall(%)	71	97	66	80	65	90	52	84	96	80
F1-score(%)	73	94	66	82	64	80	51	87	92	86

Table 2: Precision, Recall, F1-score and time spending for training classifier of each label.

When using OVO, the time needed for each class is around 3s, the model still performs well on the labels of 1, 3, 7, 8 and 9, but we also have an improvement on 0, 4, 5. Also the recall of label 6 is increased .
Figure 2 below demonstrates the confusion matrix of the model OVO of logistic regression. Compared with the baseline method, the true positive of label 0 and 5 increased a lot, this model looks like not classifying labels 0 and 3 is not that difficult any more, the FP to label 3 of label 6 in obviously declined. However, it sometimes still will label 6 as 2 .



	0	1	2	3	4	5	6	7	8	9	All
0	135	0	0	15	0	0	25	0	3	0	178
1	5	175	2	8	0	0	1	0	0	0	191
2	6	1	140	1	32	0	28	0	1	1	210
3	11	2	1	162	10	1	4	0	0	0	191
4	1	0	44	4	135	0	27	0	1	0	212
5	0	0	0	1	0	156	0	22	0	35	214
6	31	1	25	9	31	0	101	0	2	0	200
7	0	0	0	0	0	12	0	179	0	7	198
8	1	1	1	3	1	5	9	3	194	1	219
9	0	0	0	0	0	0	0	10	1	176	187
All	190	180	213	203	209	174	195	214	202	220	2000

Figure 2:OVO

When we compare the best performances for each optimisation method (GD, Momentum and TNC), we can see that the choice of optimisation method does not affect our accuracy that much. Even though the 78.5% accuracy of GD is

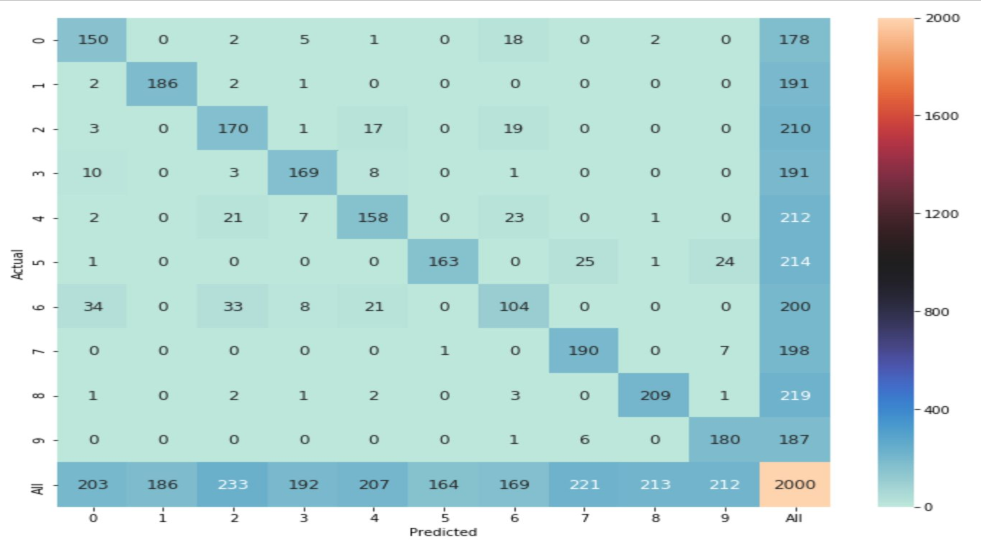
slightly higher than the other two, 78.3% for momentum and 77.3% for TNC, we do not see much difference.
However, the time spending has been improved a lot using Momentum and TNC method. Considering the baseline Gradient Descent method, it costs 313s in total. More specifically, for training (0, 6) and (2, 4) pair, calculations have taken up to 24 seconds. For the rest of the pairs of classes, the calculation time is less than 10s based on the output. Using Momentum method, the time spent has decreased from 313s to 223s. Most of the calculations for the pairs have dropped to 5s. Surprisingly, when using TNC method, the time spent decreased even more to 96s and calculation time of each pair to less than 3s.

4.4 KNN

K	3	5	7	9	11	13	15
Accuracy	84.15%	83.75%	83.95%	83.55%	83.25%	83.25%	83.15%

K	37	97	157	217	277
Accuracy	82.65%	81.85%	80.90%	77.90%	77.45%

The predicted accuracy is good ranging from k=3 to 15, around 83%. While k gets larger, accuracy decreases eventually. For our computing time, we have nearly the same results of around 180 seconds, since the algorithm only takes different top k neighbors for the prediction, the calculation of the distances are the same.
Here we plot our confusion matrix(figure 3), we can see that label 6 is hard to be predicted correctly, being predicted as labels 0, 2, 4. We can see that label 6 has a low score of prediction in table 3. As of the prediction, label 5 performs very well.



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	All
0	150	0	2	5	1	0	18	0	2	0	178
1	2	186	2	1	0	0	0	0	0	0	191
2	3	0	170	1	17	0	19	0	0	0	210
3	10	0	3	169	8	0	1	0	0	0	191
4	2	0	21	7	158	0	23	0	1	0	212
5	1	0	0	0	0	163	0	25	1	24	214
6	34	0	33	8	21	0	104	0	0	0	200
7	0	0	0	0	0	1	0	190	0	7	198
8	1	0	2	1	2	0	3	0	209	1	219
9	0	0	0	0	0	0	1	6	0	180	187
All	203	186	233	192	207	164	169	221	213	212	2000

Figure 3: K=7

Label	0	1	2	3	4	5	6	7	8	9
Precision(%)	85	98	81	88	73	77	53	95	96	97
Recall(%)	72	100	71	89	78	99	65	86	99	85
F1-score(%)	78	99	76	89	76	87	58	90	97	90

Table 3: Precision, recall, F1-score of the KNN model where K=7

Here we take a look at our confusion matrix(figure 3) and our precision, recall, F1-score table(table 3). Label 6 has a low score on precision with only 53%, and we can see from figure 1, only 104 data points are predicted correctly, others spreaded across labels 0, 2, and 4. For the recall rate, labels 1, 5, 8 perform very well, which we can see from the rows of the labels in the figure. Label 6 on the other hand still performs poorly for the recall rate. Finally we look at our F1-score, it's not surprising that label 6 has the lowest score. Labels 0, 2, 4 doesn't perform too well, mainly because of its low score in the recall rate, affected by the false negative predictions of label 6, which we can see from the column of label 6 in the figure. To sum up, besides label 6, the model performs fairly well.

Comparing to the other classifiers we implemented, OVA and OVO, labels 0, 2, 4, 6 perform not so well, having label 6 being the worst in all three classifiers. Computing time for KNN is about two times or more than the other two, since we do a lot of calculation in the algorithm.

5 Extensive Analysis

5.1 PCA percentage retained

When we keep more information, this means there will be higher variance retained. For this data, higher variance of the information, the accuracy and the other metrics will be larger. But when the variance(65%) is too low, all three models are similarly low in accuracy. However,more information means more time .To balance the time and accuracy, we find the most suitable value 95% here.

5.2 Learning rate

We can see the following graph(figure 4) that the best learning rate is 0.1, where the validation loss is the lowest. When test from learning_rate=e**-5 to learning_rate=0.1 the line descent distinctly when we use =1000,p value =e**-3. and the line goes up slightly after the learning_rate bigger than 0.1.

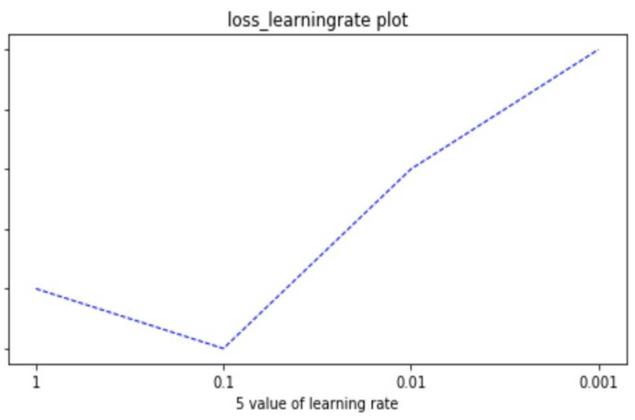


Figure 4: Loss-learning rate plot

5.3 Iteration

Iteration affects the running time significantly, and it also influences the accuracy and the other metrics. The following table illustrates when iteration number n increases, the time spent and accuracy both rise for the OVO and OVA model. Comparing to the OVO, the time increases significantly at n=5000 for the OVA model.

OVO/OVA	n=1000	n=2000	n=3000	n=4000	n=5000
Time(s)	91/70	93/70	95/73	97/75.3	110/145
Accuracy(%)	75.63/ 73.4	76.24/ 73.45	77.5/ 73.6	77.98/ 73.85	78.1/ 73.96

Table 4: Time and accuracy for OVO, OVA models at different n.

6 Discussion

For solving image classification problems, KNN, Logistic Regression, Naive Bayes, and SVM can be chosen and implemented as most effective models. However, there are differences between them and classifier performance varies depending on how data is being considered and the quality of data. We cannot simply say there is a best classifier for our data before consideration and testing. The two classical models, Naive Bayes(NB) and Logistic Regression(LR) for Generative Modeling(GM) and Discriminative Modeling(DM) are totally different when building models. The way Naive Bayes algorithm do the classification is based on Bayes' Theorem. It models the distribution of each feature, and then "naively" assume they are independent and multiplying them altogether to get the joint probability distribution. Therefore, lots of calculations and massive training data is required to implement Naive Bayes algorithm. On the other hand, Logistic Regression directly builds the model using P(Y|X), which saves us from too many calculations. Logistic Regression also makes an assumption that the model is linear. According to Andrew Ng and Michael Jordan's work published on NIPS, concerning one generative-discriminative pair, DM has lower asymptote error than GM(Andrew Ng and Michael Jordan, 2012). Based on all these research, since we do not always know if the features are independent or not, LR could be more reliable than NB based on the quality of our dataset.

SVM is a feasible alternative for LR. Based on research, SVM can outperform LR when the boundaries between classes are not linear, and then SVM can use the kernel trick to a characteristic space on which data is linearly separable (Salazar&Velez, 2012). Furthermore, even though SVM was originally created and developed as a binary classifier, Crammer-Singer formulation for multiclass classification is popular and frequently used when facing multiclass problems.

However, non-linear SVM ($O(N^2k)$) could be computationally more expensive than LR ($O(N)$) where k is the number of support vectors.
Meanwhile, KNN is also a simple and effective way to classify data, but we need to always test the best k value. Picking a wrong k-value will lower the model accuracy. An additional drawback is that KNN does not give you any idea of important statistics of the data. KNN has a time complexity of $O(Nd)$ where d is the dimension of each sample. It will be relatively slow since it requires large memory for storing training data for prediction.
As for our data set we need to build the model manually, we chose Logistic Regression and KNN which are simpler to implement. Our KNN model reached around 84% accuracy, which outperforms two LR models with around 78% accuracy.

7 Conclusion and future work

7.1 Conclusion based on the results

The aim of this project is to train the machine learning model to classify the 10 kinds of images(labels). This includes the following 5 steps:
Step 1: Download and pre-process the data, which includes joining and normalization of the data and do the feature selection.
Step 2: Build different classification models, here we use logistic regression and KNN.
Step 3: Use the training data to train the model and separate 2000 data points as the validation data, which is used to check whether the model is overfitting or underfitting. Then adjust the parameters with the validation result.
Step 4: Do more extensive discussions to adjust the parameter. Also use different optimisation methods to record and contrast the metrics and computing time to get the model with the fittest optimisation method.
Step 5: Compare different models by testing with the testing data.
Our final result is to choose KNN with k=7, and accuracy at 84.78%, the precision, recall and F1 score are 75.8%, 84.4%, 84%.

7.2 Future work

We find that there are some meaningful work that could be done in the future. First, for our dataset, if calculate the accuracy for each label, we can find that label 6 has a relative low accuracy, hence we can implement a different model when training data points labeled as label 6. Second, we can draw more line chart to show the change of loss through iteration time in gradient descent with more optimisation methods to adjust the relative parameters to have a higher accuracy. Last but not least, we can use more models like SVM, CNN, MLP for more comparison, and try to find the fittest model for this data.

8 Appendix

1. Training and test dataset: images_training.h5, images_testing.h5, labels_training.h5 and lables_testing 2000.h5
2. Confusion Matrix: The confusion matrices generated by using baseline system and OVO system has been saved in this folder.
3. Hardware and software: Macbook Pro(2016), Processor: 2.6GHz Intel Core i7, Memory: 16GB 2133MHz, Graphics: Radeon Pro 450 2048 MB, Intel HD Graphics 530 1536 MB, Software: macOS Sierra 10.12.3

9 References

Alex Smola and S.V.N. Vishwanathan(2008). *Introduction to machine learning*. Retrieved from http://alex.smola.org/drafts/thebook.pdf

Diego Alejandro Salazar and Jorge Ivan Velez(2012). *Comparison between SVM and Logistic Regression: Which One is Better to Discriminate?* Retrieved from http://www.kurims.kyoto-u.ac.jp/EMIS/journals/RCE/V35/v35n2a03.pdf

Ng, A. Y., and Jordan, M. I. (2001). *On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes*. In NIPS (pp. 841–848).