Writing Assignment Report

Nicola Greco (2327775) and Ivo Borkus (6746713)

2023-12-07

Contents

0-Introduction

The Modified National Institute of Standards and Technology (MNIST) dataset serves as the foundation for our analysis. It is a renowned collection of 42,000 handwritten digits extensively utilized in the field of machine learning for image processing tasks.

The primary objective of our analysis was to examine various machine learning models' ability to accurately predict which digit had been written based on these pixel images and other derived features. This investigation will encompass different models such as Logistic Regression, LASSO regularization, and Support Vector Machines (SVM).

One noteworthy aspect throughout our modeling phase involves tuning hyperparameters related to model complexity using cross-validation (cv). These parameters control overfitting versus underfitting trade-off

- too much complexity may lead to overfitting where our model memorizes noise instead of generalizing patterns whereas too little can result in underfitting where relevant trends are missed out.

REF(https://docs.ultralytics.com/datasets/classify/mnist/#extended-mnist-emnist).

1-The data exploration

In this section, we conducted an exploratory analysis of the MNIST dataset to understand its characteristics and structure.

1.1-Structure of the data set

As stated before the data set contains 42000 samples of handwritten digits. The digits ranges from 0 to 9, with each pixel value varying between 0 (white) and 255 (black). The size of each image is standard at 28x28 pixels, hence each instance in this dataset holds 784 feature values (pixels), with an additional column allocated for class labels:

```
##   label pixel201 pixel202 pixel203 pixel204 pixel205 pixel206 pixel207 pixel208
## 1     1        0        0        0        0        0        0        0        0
## 2     0        0        0       72      254      254      254      254      254
## 3     1        0        0        0        0        0        0        0        9
```

Example of a digit, or 1 row of data (excluding the label):

1.2-Missing data and useless pixels

The second step in our data exploration was to check for missing values within the dataset. We found that no NA or null values present in the dataset. This makes sense as pixels without any colour are marked as 0 expression and no labels are missing.

Having established that there was no missing information to deal with, we moved on to summarize the statistics for each pixel. The statistical measures used included minima, maxima, first quartile (25th percentile), second quartile (median), third quartile (75th percentile), and mean. These statistics provided insights into variation among pixel intensities and their potential discriminative power.

```
##         Min. 1st Qu. Median     Mean 3rd Qu. Max.
## pixel93    0       0      0 2.292857       0  255
## pixel94    0       0      0  3.768381       0  255
## pixel95    0       0      0  5.713881       0  255
## pixel96    0       0      0  7.751238       0  255
## pixel97    0       0      0 10.048857       0  255
## pixel98    0       0      0 12.067738       0  255
```

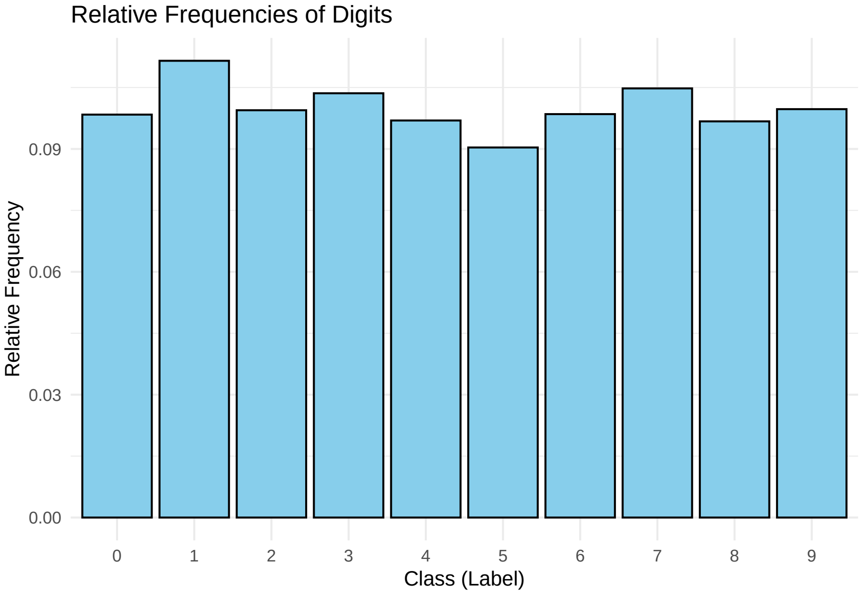
We then moved on to investigate if there were any pixels that could be considered useless or redundant for our predictive modeling. We identified these as pixels with zero variance across all images, meaning they held identical values across all instances in the data set. Such features would not contribute any useful information for differentiating between digit classes and hence can be safely removed from our feature set.

This is an example of some of the pixel that we defined Useless:

```
##         Min. 1st Qu. Median Mean 3rd Qu. Max.
## pixel23    0       0      0    0       0    0
## pixel24    0       0      0    0       0    0
## pixel25    0       0      0    0       0    0
## pixel26    0       0      0    0       0    0
## pixel27    0       0      0    0       0    0
## pixel28    0       0      0    0       0    0
```

We so identified 76 such "useless" pixels which were subsequently excluded from further analyses.

Another important aspect of data exploration is understanding class distribution. Our task is a multi-class classification problem, where we are trying to classify each image into one of ten classes (digits 0-9). A skewed class distribution can often lead to biased models that tend towards predicting majority classes while performing poorly on minority ones.



Plotting relative frequencies revealed that our classes are fairly balanced with '1' being slightly more abundant than others. However, it's worth noting that even if we naively predicted every instance as belonging to this majority class ('1'), our accuracy would only be:

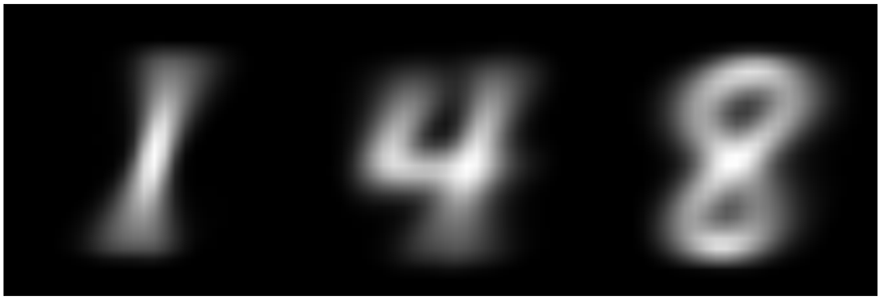
```
## 11.15238 %
```

This gives us a baseline accuracy that any subsequent model should aim to surpass.

1.2 - Average digits

To gain a deeper understanding of our data and examine any visual patterns that may exist, we decided to calculate the mean pixel value for each label. In other words, we aimed to create an ‘average’ digit representation for each label in our dataset. This involved grouping the dataset by labels and then computing the mean of each column within these groups.

Following this computation process, we plotted these ‘average’ digit representations to visually ascertain their semblance with actual handwritten digits. Here are few examples:



The above plots represent average images corresponding to labels ‘1’, ‘4’, and ‘8’. On close observation of these images alongside their corresponding labels gives us interesting insights into how different parts of a digit may contribute towards its identification.

While we realize that some pixels clearly indicate the presence of specific digits due to their consistent appearance across different samples (as they appear darker in the average image), there are also pixels that do not contribute significantly towards identifying a particular digit (as they appear lighter).

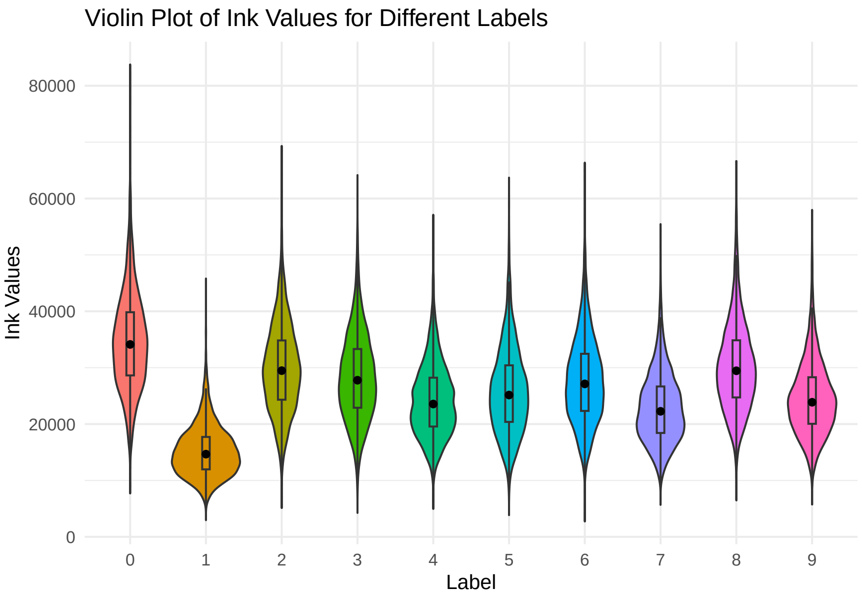
By performing this analysis at an early stage in our project gave us valuable insights about feature importance which could guide our further analysis.

1.3- New features

Using the knowledge we gained from the exploratory analysis we concluded that we could build three relatively simple features to help aid the multinomial logistic regression model in correctly classifying digits.

1.3.1 - Ink per digit The first feature we introduce is called ‘ink’, which quantifies the sum of all pixel values in an image. The intuition behind this is the following: Different digits may inherently require varying amounts of ink due to their shapes and structures. One example of this is the digit 1 and 0, a one will most likely be represented as a single dash, whereas the 0 might use significantly more of the image when writing it in a circle.

We plotted violin plots to visualize the distribution of ink values across different classes:



From above plot it was clear that there are significant differences in the ink distribution between some classes (e.g., classes '1' and '0'), but others look very similar. To further investigate this aspect, we computed mean and standard deviation of ink feature for each class:

```
##    label mean.ink   sd.ink
## 1      0 34632.41 8462.916
## 2      1 15188.47 4409.932
## 3      2 29871.10 7653.922
## 4      3 28320.19 7574.975
## 5      4 24232.72 6375.416
## 6      5 25835.92 7527.595
## 7      6 27734.92 7531.413
## 8      7 22931.24 6169.042
## 9      8 30184.15 7778.354
## 10     9 24553.75 6466.003
```

This table gives us the average amount of 'ink' and its standard deviation for each digit. By comparing these values across digits, we can infer which digits may be more distinct or similar in terms of their 'ink' usage.

The insight derived from this analysis is crucial as it indicates that our newly created feature - 'Ink', could help distinguish between different digits to a certain extent.

1.3.2 Principle Component Analysis (PCA) The second feature that can be supplied to our logistic regression model is a principle component of the PCA dimension reduction algorithm. After several iterations of testing various transformations and computed attributes from the raw pixel data, we identified the first principal component (PC1) as a promising feature. The first principal component captures most of the variance in our data and helps us understand how each digit is written.

The intuition behind this choice is that the projection of a point on the PC1 represents a linear combination of our original variables (the pixels), which maximizes the total variance in our dataset. Hence, PC1 holds significant information about patterns or structures in these images that might be vital for distinguishing one digit from another.

To compute this new feature, we first carried out Principal Component Analysis (PCA) using R's `prcomp` function form the packages *stats* with 'scale = TRUE' and default settings for the other paramaters. We then projected our original data onto PC1 through a dot product operation between each point and PC1:

```
# Perform Principal Component Analysis (PCA)
pca_result <- prcomp(features, center = TRUE, scale. = TRUE)

projection_1pc <- as.vector(as.matrix(features) %*% pca_result$rotation[, 1])
```

2-Machine learning analysis.

2.1 - Logit on Ink only as predictor

Our first approach to model fitting was a simple logistic regression using the 'ink' feature as the predictor variable.

The data was prepared by scaling the 'ink' column in our dataset. The purpose of this scaling was to standardize our predictor variable so that it has a mean of 0 and a standard deviation of 1, which helps in ensuring consistent interpretation of coefficients across different scales and to ensure that each feature contributes proportionally to the final prediction:

After scaling, we fit a multinomial logistic regression model using only the 'ink' feature. We used the *multinom()* function form the packages *nnet*.

```
# Fit multinomial logistic regression model using only 'ink' feature
multi.logit <- multinom(label ~ ink.scaled, data = mnist.dat.ink)
```

```
## # weights:  30 (18 variable)
## initial  value 96708.573906
## iter  10 value 86968.775102
## iter  20 value 86672.373390
## final  value 86623.202362
## converged
```

This provided us with estimated parameters for each class in relation to our predictor variable:

```
## Call:
## multinom(formula = label ~ ink.scaled, data = mnist.dat.ink)
##
## Coefficients:
##   (Intercept) ink.scaled
## 1  -1.7665025 -4.4375015
## 2   0.4225402 -0.5996883
## 3   0.5399180 -0.8297827
## 4   0.4729364 -1.5544628
## 5   0.4440854 -1.2464037
## 6   0.5089846 -0.9221906
## 7   0.4666119 -1.8329584
## 8   0.3754043 -0.5555656
## 9   0.5144041 -1.4898559
##
## Residual Deviance: 173246.4
## AIC: 173282.4
```

Interestingly, when we observed parameters estimated by this model, it indicated certain digits are more likely given higher or lower amounts of ink usage. However, considering that handwritten digits can vary greatly in size and thickness between individuals and even different instances from the same individual, we did not expect this model to be highly accurate.

Further analysis was conducted on pairs of digits with notably different and similar distributions of ink usage respectively. We hypothesized that a logistic regression model would perform better on a simpler task of distinguishing between pairs with diverse ink distribution such as 1-8 or 4-0 than those with similar distributions like 3-8 or 9-4.

In essence, what we did here was creating sub-datasets containing only two types of digits at a time (e.g., only digits ‘1’ and ‘8’), then training separate logit models on these smaller datasets:

```
two.digits.dataset <- mnist.dat.ink[mnist.dat.ink$label %in% c(1,8),]

model <- multinom(label ~ ink.scaled, data = two.digits.dataset)
```

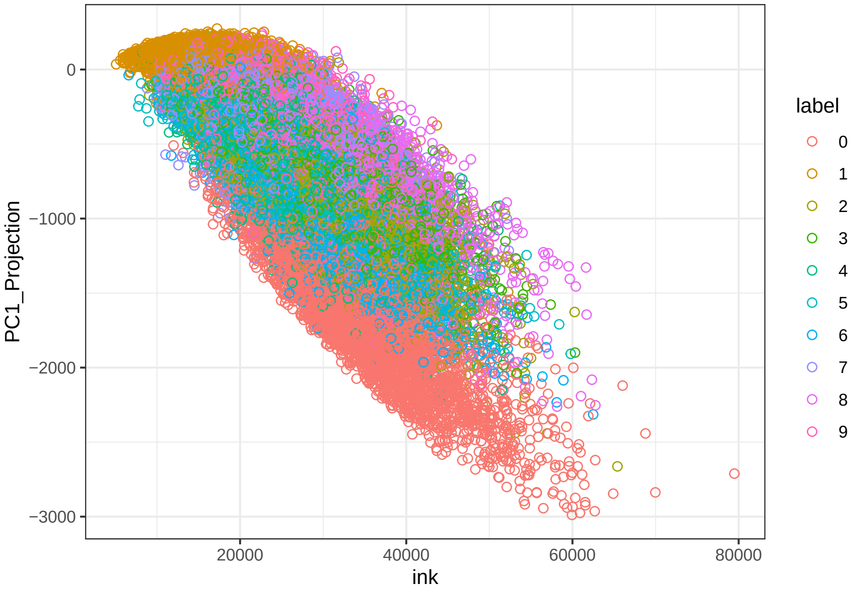
2.2 - Logit with Ink and additional feature as predictiors

To improve upon our previous model’s performance, we decided to introduce an additional predictor variable derived from Principal Component Analysis (PCA).

We trained a new multinomial logistic regression model using both the ‘ink’ and ‘1PC_Projection’. The rationale behind this combination was that while ink quantifies amount of writing, the PCA feature could potentially capture patterns in how digits are written. Thus, together they might provide a more comprehensive description of each digit.

```
##   label   ink PC1_Projection ink.scaled
## 1     1 16649      -122.9040 -1.0983307
## 2     0 44609     -2318.1255  2.1194666
## 3     1 13425       112.9645 -1.4693671
## 4     4 15025      -539.8415 -1.2852299
## 5     0 51093     -2734.8879  2.8656825
## 6     0 23061      -832.1524 -0.3604009
```

Here we plot the ink value on the y axis and the PC1 projection on the x.



We noticed that while some labels appeared clustered together when plotted against these two features ('ink' on y-axis vs 'PC1_Projection' on x-axis), they were not linearly separable. Hence it was anticipated that the prediction ability of this dual-feature model may not be significantly superior to the single-feature one. Again we used the *multinom()* function form the packages *nnet*:

```
# Fit multinomial logistic regression models using the first principal components
model <- multinom(label ~ ink.scaled + PC1_Projection, data = mnist.dat.ink)
```

2.3 - Multinomial Logistic Regression with Lasso regularization

In this section, we applied the Multinomial Logistic Regression with Lasso regularization to our data set. Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function that the algorithm optimizes. The objective was to develop a predictive model that can accurately classify handwritten digits using the raw pixel values themselves as features.

Before starting, we prepared the data by creating a training and testing split. It's important to mention that we eliminated pixels that have zero variance since they would not contribute to any predictive value in our models due to their constant nature. This procedure helps us reduce dimensionality and computational complexity. We set the seed for reproducibility: *set.seed(123)*.

The split was done by sampling 5000 examples from the dataset for training purposes, while the rest was reserved for model validation:

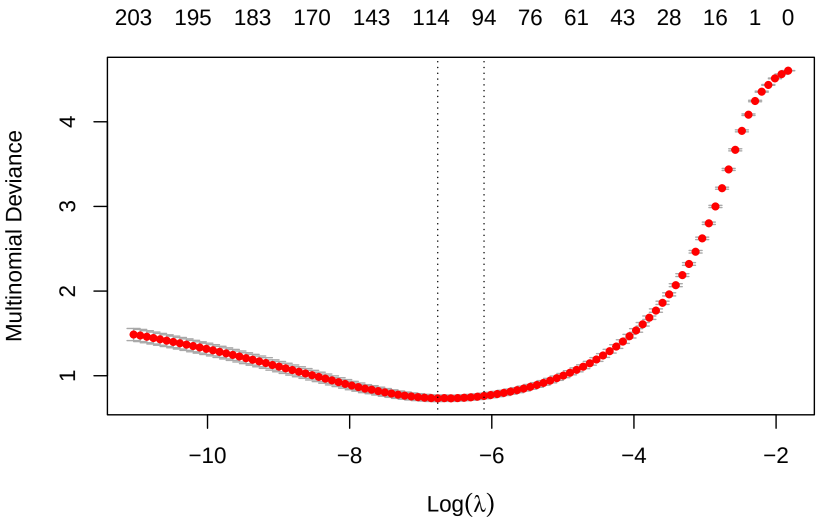
Next, we applied Cross-Validation (CV) on our training data to tune hyperparameters in our model. In specific, CV aids in reducing overfitting, improving algorithm performance and helping us choose the most optimal parameters for our model.

The function `cv.glmnet()` from `glmnet` package was used for this purpose where alpha = 1 implies application of pure lasso regression (L1 penalty) and no ridge regression (L2 penalty). Here 'alpha' is a parameter for elastic net mixing which combines properties of both lasso and ridge penalties.

```
cv.logit <- cv.glmnet(x = predictors, y = response,
                                            family = "multinomial", nfolds = 15, alpha = 1)
```

This tuning method returns the optimal value of lambda (the tuning parameter that controls the overall strength of the penalty) that minimizes the cross-validated error.

Our result plot shows a curve indicating how model performance varies with different values of lambda.



The best value of 'lambda' is chosen based on two rules: 'min' and '1se'. The former represents absolute minimum, while the latter allows for model simplicity, where it picks maximum lambda such that error is within one standard error of the minimum.

The cv.glmnet function gives us both these options and we chose to proceed with '1se' as it provides a simpler model by reducing complexity which helps avoid overfitting.

```
##
## Call:  cv.glmnet(x = predictors.some.pixel, y = response.some.pixel,      nfolds = 10, family = "mul
##
## Measure: Multinomial Deviance
##
##       Lambda Index Measure     SE Nonzero
## min 0.001159    54  0.7327 0.0379     114
## 1se 0.002222    47  0.7611 0.0327      94
```

Interestingly, only 94 coefficients were non-zero. This means only 94 pixels are used for making predictions which demonstrates Lasso's feature selection ability by driving irrelevant features' coefficients to zero and hence retaining only significant predictors.

2.4 SVM

Finally, we trained different Support Vector Machine (SVM) models with various kernels and parameters using cross-validation to find optimal hyperparameters.

To find an optimal model configuration for our task, we used the 'tune.svm()' function from the 'e1071' package in R. This function performs a grid search over specified parameter ranges, training an SVM for each combination and evaluating its performance using cross-validation.

For radial basis function (RBF) kernel, two key parameters were tuned: gamma and cost. Gamma controls the influence of individual training samples on shaping the decision boundary while cost determines how much penalty should be applied for misclassification during training.

For the SVM model with linear kernel the only parameter to be tuned was cost, with the same effect that it has in the radial.

Finally, we explored the polynomial kernel that allows our SVM model to create polynomial decision boundaries. We fine-tuned both ‘degree’, which determines the degree of the polynomial used for creating decision boundary and ‘cost’ as earlier.

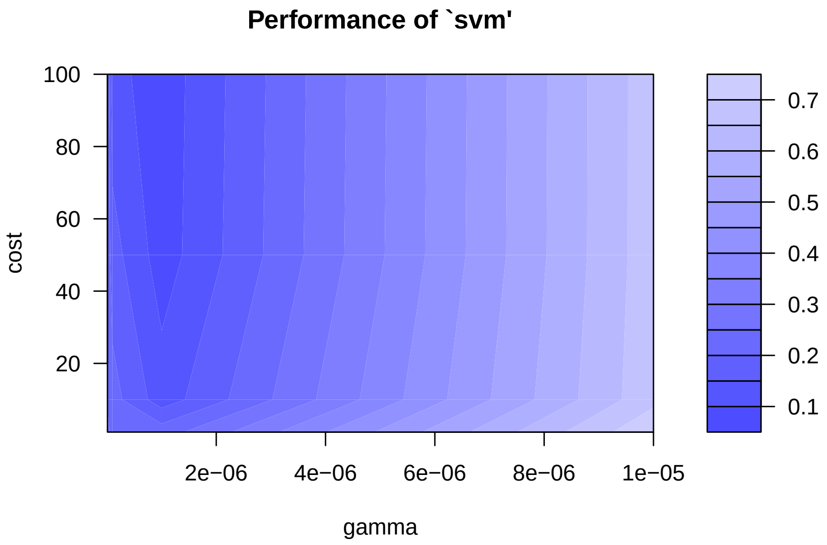
```
# Tune different SVM models with different kernels and parameter ranges:

# Radial kernel SVM models
cv.svm.radial <- tune.svm(label ~ ., data = train.mnist.dat, kernel = "radial"
                          gamma = c(10^-8,10^-7,10^-6,10^-5), cost = c(1,10,50,100))


# Linear kernel SVM
cv.svm.linear <- tune.svm(label ~ ., data = train.mnist.dat,
                          kernel = "linear", cost = 10^(-1:2))


# Polynomial kernel SVM
cv.svm.polynomial <- tune.svm(label ~ ., data = train.mnist.dat,
                          kernel = "polynomial", degree = 2:3, cost = 10^(-1:2))
```

CV for radial: The graph displays the error across different combinations of cost and gamma values resulted form the CV of SVM with radial kernel. It provides visual intuition about the parameter space and how model performance changes with varying parameters:

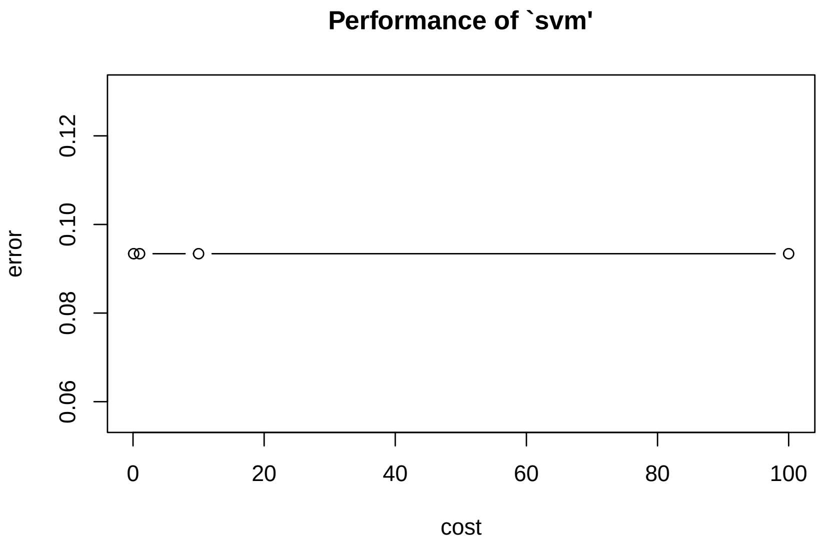


From the table below that showcases the performances per hyperparameter combination:

```
##    gamma cost  error dispersion
## 1  1e-08    1 0.2664 0.32471362
## 2  1e-07    1 0.2298 0.34392305
## 3  1e-06    1 0.2266 0.34564471
## 4  1e-05    1 0.7426 0.19136016
## 5  1e-08   10 0.2420 0.33755855
## 6  1e-07   10 0.2190 0.34947866
## 7  1e-06   10 0.1232 0.13443611
## 8  1e-05   10 0.6860 0.30199485
## 9  1e-08   50 0.2388 0.33935782
## 10 1e-07   50 0.1698 0.24479370
## 11 1e-06   50 0.0746 0.03400719
## 12 1e-05   50 0.6818 0.31080748
## 13 1e-08  100 0.2418 0.33775922
## 14 1e-07  100 0.1184 0.13644958
## 15 1e-06  100 0.0704 0.02615849
## 16 1e-05  100 0.6818 0.31080748
```

We so identified optimal gamma as 1e-06 and cost as 100 which yielded the smallest cross validation error of approximately 0.0704.

CV for linear: The process is repeated for linear kernel SVM model. Hyperparameters error plot:



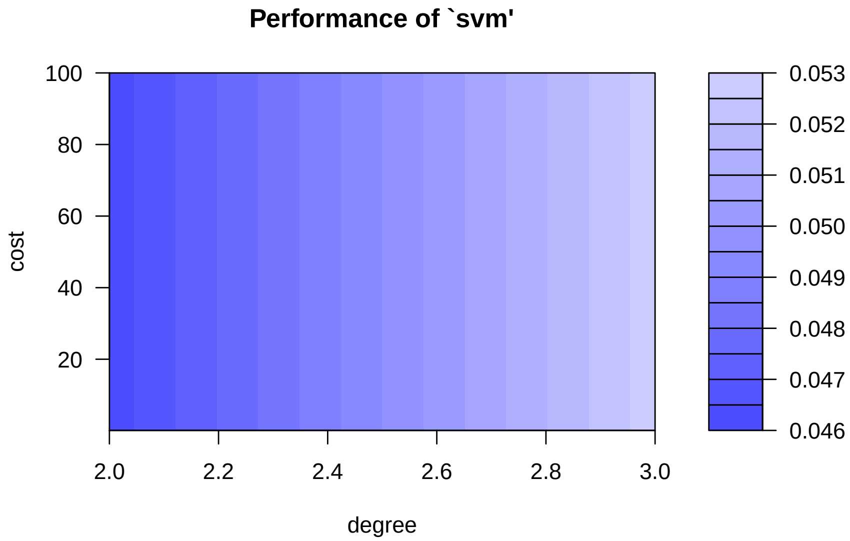
For linear kernel-based SVMs, it seems that changing the cost did not affect model performance as indicated by constant cross-validation errors in plot above. We can see it clearly by plotting the performances table for each hyperparameter value:

```
##    cost  error dispersion
## 1   0.1 0.0934 0.01454648
## 2   1.0 0.0934 0.01454648
## 3  10.0 0.0934 0.01454648
## 4 100.0 0.0934 0.01454648
```

So we select a cost value of 0.1 with an associated performance score of approximately 0.0934. We accepted the default choise of the *tune.svm()* function that automatically choose the lowest parameters in case there is no clear best option.

CV for Polynomial And also for polynomial kernel SVM model.

Hyperparameters error plot:



Through cross-validation hyperparameter tuning, sveral optimal degree and cost combination were found, with a performance of approximately 0.046. We so choose to select the model with degree and cost 2 and 0.1 respectively, to maintain the model as simple as possible.

```
##   degree  cost  error  dispersion
## 1      2   0.1 0.0462 0.005769652
## 2      3   0.1 0.0528 0.007004760
## 3      2   1.0 0.0462 0.005769652
## 4      3   1.0 0.0528 0.007004760
## 5      2  10.0 0.0462 0.005769652
## 6      3  10.0 0.0528 0.007004760
## 7      2 100.0 0.0462 0.005769652
## 8      3 100.0 0.0528 0.007004760
```

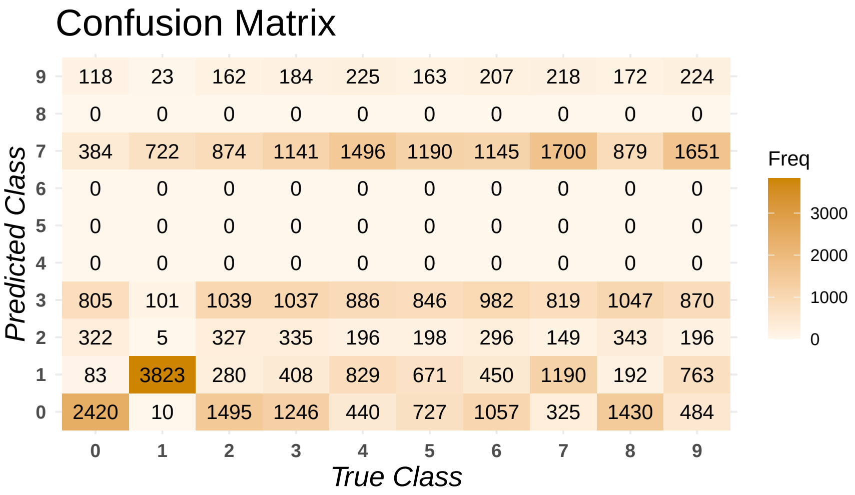
3-Discussion:

3.1 - Logit on Ink only

The first logit model trained will help us understand how much of the classification task can be done by considering solely the total ink usage in digit images.

The first step is to use our trained model to predict classes on the same dataset that we used for training. Although not typically recommended due to overfitting concerns, given that this part of the assignment only requires evaluating simplistic models, we are using complete data set for both training and evaluation

Let’s take a look at the resulting confusion matrix and other statistics provided by caret package:

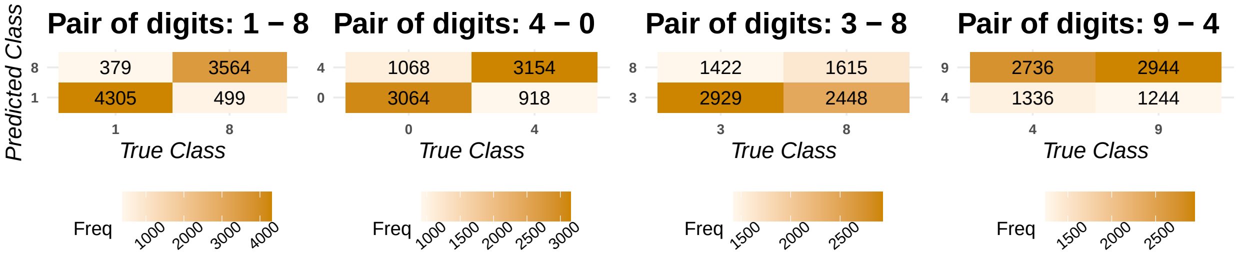


	0	1	2	3	4	5	6	7	8	9
9	118	23	162	184	225	163	207	218	172	224
8	0	0	0	0	0	0	0	0	0	0
7	384	722	874	1141	1496	1190	1145	1700	879	1651
6	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
3	805	101	1039	1037	886	846	982	819	1047	870
2	322	5	327	335	196	198	296	149	343	196
1	83	3823	280	408	829	671	450	1190	192	763
0	2420	10	1495	1246	440	727	1057	325	1430	484

Statistics	values
Accuracy	0.2269
95% CI	0.2229, 0.231

The results show that our simplistic model struggles when differentiating among digits 4-5-6 which were not predicted at all leading to low overall performance with an accuracy rate of around 22%.

To further assess our model's performance more directly, let's perform an experiment where we focus on predicting specific pairs of digits. The aim is to examine whether there are certain digit pairs that can be distinguished easily based on the ink feature alone.



Pair of digits: 1 − 8

	1	8
8	379	3564
1	4305	499

Pair of digits: 4 − 0

	0	4
4	1068	3154
0	3064	918

Pair of digits: 3 − 8

	3	8
8	1422	1615
3	2929	2448

Pair of digits: 9 − 4

	4	9
9	2736	2944
4	1336	1244

Statistics	values
Accuracy	0.8996
95% CI	0.8931, 0.9058

Statistics	values
Accuracy	0.7579
95% CI	0.7485, 0.7672

Statistics	values
Accuracy	0.5401
95% CI	0.5293, 0.5507

Statistics	values
Accuracy	0.5182
95% CI	0.5073, 0.529

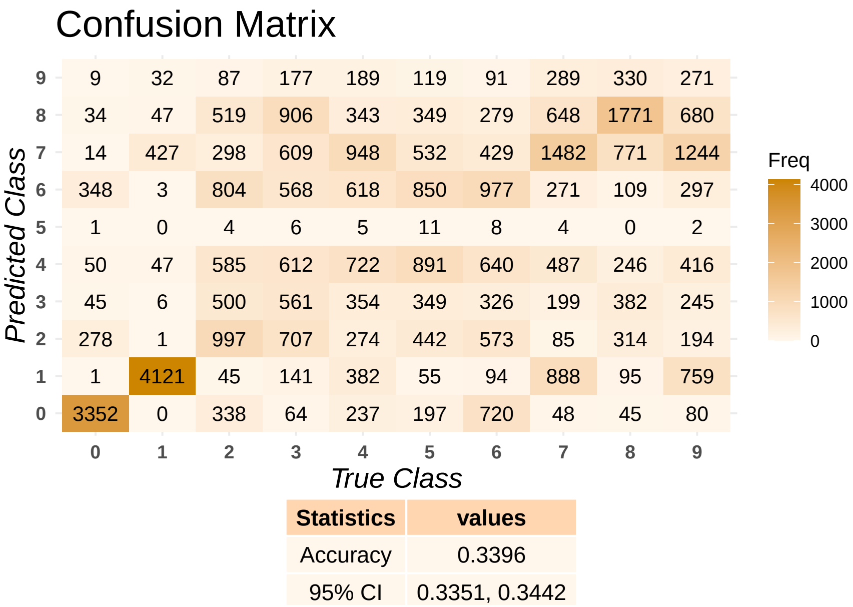
The above results reveals that certain pairs like '1' and '8', '4' and '0', have higher accuracies around ~90% and ~76% respectively suggesting that these pairs can be distinguished quite accurately with just ink usage information. However, for other digit pairs, the model does not perform well.

In conclusion, using ink as the sole feature to classify digits has its limitations but can provide some reasonable performance for certain pairs of digits. The next sections will present results from more sophisticated models that utilize multiple features.

3.2 - Logit with Ink and 1PC

The confusion matrix was then generated for this model to evaluate its predictive accuracy on each digit:

Let's take a look at the resulting confusion matrix and other statistics provided by caret package:



Confusion Matrix

Predicted Class \ True Class	0	1	2	3	4	5	6	7	8	9
9	9	32	87	177	189	119	91	289	330	271
8	34	47	519	906	343	349	279	648	1771	680
7	14	427	298	609	948	532	429	1482	771	1244
6	348	3	804	568	618	850	977	271	109	297
5	1	0	4	6	5	11	8	4	0	2
4	50	47	585	612	722	891	640	487	246	416
3	45	6	500	561	354	349	326	199	382	245
2	278	1	997	707	274	442	573	85	314	194
1	1	4121	45	141	382	55	94	888	95	759
0	3352	0	338	64	237	197	720	48	45	80

Statistics	values
Accuracy	0.3396
95% CI	0.3351, 0.3442

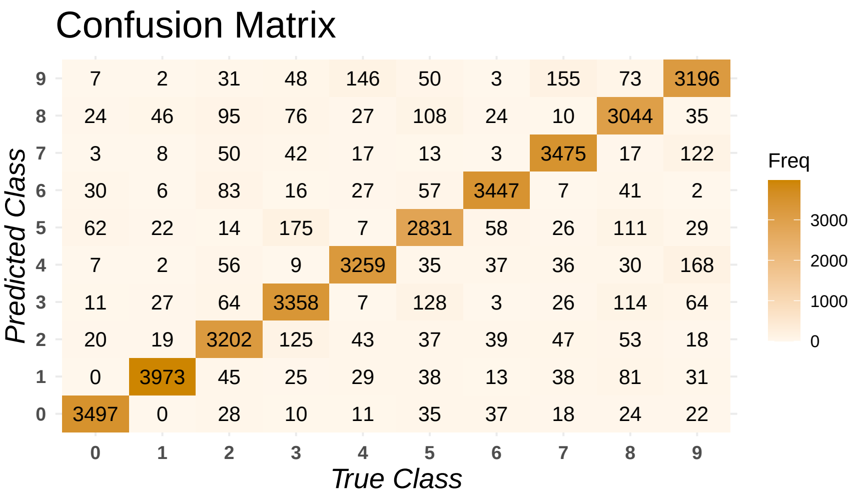
We observed improved overall accuracy compared to the previous models that only used 'Ink' as a single feature; suggesting that combining multiple features can improve classification outcomes in our dataset.

Interestingly, we found that the digits '0' and '1' had the highest sensitivity and positive predictive rate. This observation might be attributed to the unique ink distribution in these digits, distinct from others, as was seen with the 'Ink' only model.

In summary, utilizing both the total ink used and patterns of writing obtained via PCA served to increase our model's predictive power. However, there is still room for improvement; performance remained sub-optimal for several digits. We will explore other techniques, such as regularization and support vector machines (SVM), in subsequent sections to see if these methods can further enhance our models' performances.

3.3 - Multinomial Logistic Regression with Lasso regularization

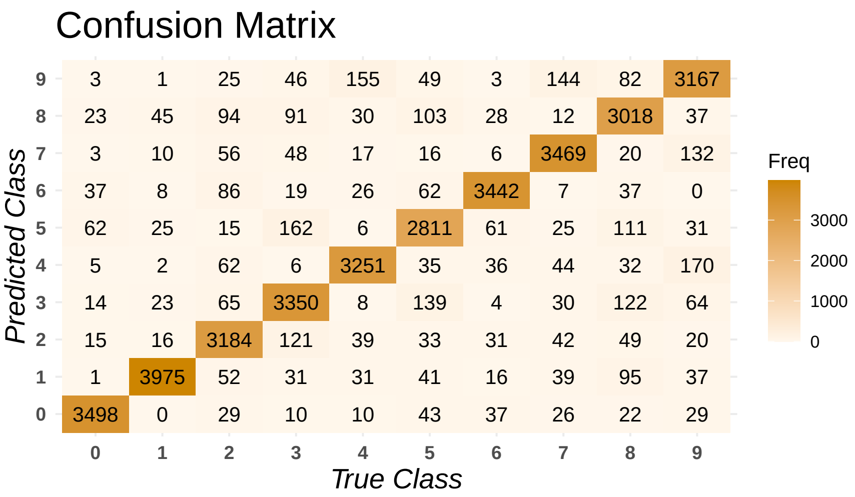
We proceeded to evaluate the performance of our multinomial logistic regression model using Lasso regularization. The two critical hyperparameters in this case were the lambda values: 'lambda.min' (minimal) and 'lambda.1se' (simplest). We assess our model's classification performance using confusion matrices for both lambda values starting with 'lambda.min':



	0	1	2	3	4	5	6	7	8	9
9	7	2	31	48	146	50	3	155	73	3196
8	24	46	95	76	27	108	24	10	3044	35
7	3	8	50	42	17	13	3	3475	17	122
6	30	6	83	16	27	57	3447	7	41	2
5	62	22	14	175	7	2831	58	26	111	29
4	7	2	56	9	3259	35	37	36	30	168
3	11	27	64	3358	7	128	3	26	114	64
2	20	19	3202	125	43	37	39	47	53	18
1	0	3973	45	25	29	38	13	38	81	31
0	3497	0	28	10	11	35	37	18	24	22

Statistics	values
Accuracy	0.899513513513513
95% CI	(0.8964, 0.9025)

The output reveals a relatively high overall accuracy (89.95%). Following this, we evaluated performances with ‘lambda.1se’:



	0	1	2	3	4	5	6	7	8	9
9	3	1	25	46	155	49	3	144	82	3167
8	23	45	94	91	30	103	28	12	3018	37
7	3	10	56	48	17	16	6	3469	20	132
6	37	8	86	19	26	62	3442	7	37	0
5	62	25	15	162	6	2811	61	25	111	31
4	5	2	62	6	3251	35	36	44	32	170
3	14	23	65	3350	8	139	4	30	122	64
2	15	16	3184	121	39	33	31	42	49	20
1	1	3975	52	31	31	41	16	39	95	37
0	3498	0	29	10	10	43	37	26	22	29

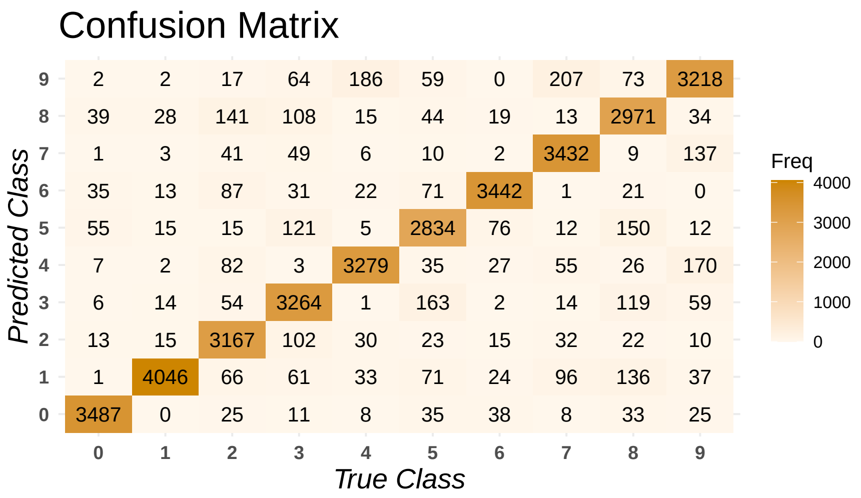
Statistics	values
Accuracy	0.896351351351351
95% CI	(0.8932, 0.8994)

With ‘Lambda.se’, which yields a simpler model relative to ‘Lambda.min’, we still manage to retain a comparable accuracy rate (89.64%). This demonstrates Lasso’s capability in performing feature selection, providing us with a model that’s less complex without significantly sacrificing classification performance.

3.4 - SVM

CV results and comparison between different kernel:

Results for radial: Applying the best parameters (gamma: 1e-06, cost: 100) and running predictions on our test data set, a confusion matrix was generated to understand how well our model performs:

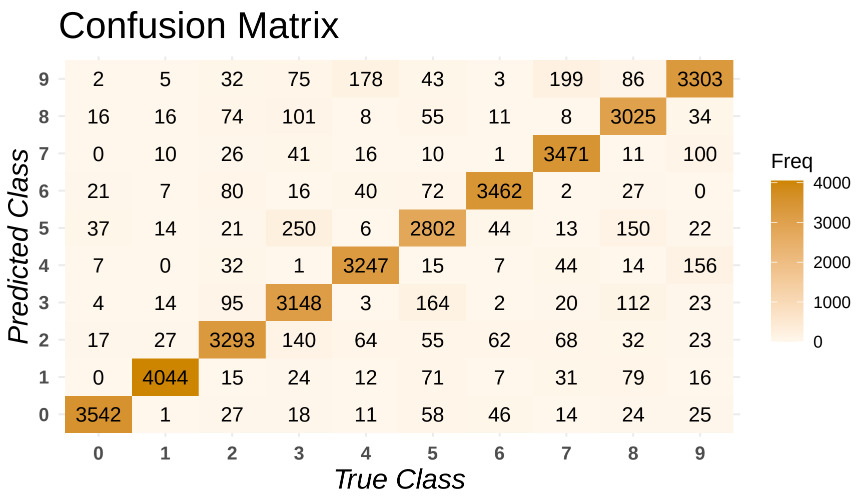


	0	1	2	3	4	5	6	7	8	9
9	2	2	17	64	186	59	0	207	73	3218
8	39	28	141	108	15	44	19	13	2971	34
7	1	3	41	49	6	10	2	3432	9	137
6	35	13	87	31	22	71	3442	1	21	0
5	55	15	15	121	5	2834	76	12	150	12
4	7	2	82	3	3279	35	27	55	26	170
3	6	14	54	3264	1	163	2	14	119	59
2	13	15	3167	102	30	23	15	32	22	10
1	1	4046	66	61	33	71	24	96	136	37
0	3487	0	25	11	8	35	38	8	33	25

Statistics	values
Accuracy	0.8957
95% CI	0.8925, 0.8988

From above, we can infer that our model performed quite well with an overall accuracy of around 89%. Sensitivity and specificity metrics also indicate a good class-wise performance by our model.

Results for linear: Confusion Matrix on Test set:

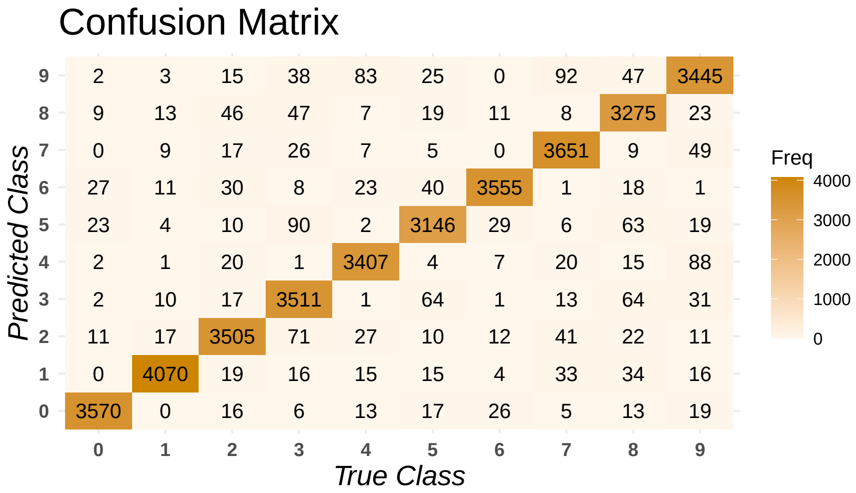


	0	1	2	3	4	5	6	7	8	9
9	2	5	32	75	178	43	3	199	86	3303
8	16	16	74	101	8	55	11	8	3025	34
7	0	10	26	41	16	10	1	3471	11	100
6	21	7	80	16	40	72	3462	2	27	0
5	37	14	21	250	6	2802	44	13	150	22
4	7	0	32	1	3247	15	7	44	14	156
3	4	14	95	3148	3	164	2	20	112	23
2	17	27	3293	140	64	55	62	68	32	23
1	0	4044	15	24	12	71	7	31	79	16
0	3542	1	27	18	11	58	46	14	24	25

Statistics	values
Accuracy	0.901
95% CI	0.8979, 0.904

The overall accuracy of the linear SVM model was approximately 90%. The specificity and sensitivity scores also indicate a good performance across different classes.

Results for Polynomial: Below is the confusion matrix for SVM with polynomial kernel function:



Predicted Class \ True Class	0	1	2	3	4	5	6	7	8	9
9	2	3	15	38	83	25	0	92	47	3445
8	9	13	46	47	7	19	11	8	3275	23
7	0	9	17	26	7	5	0	3651	9	49
6	27	11	30	8	23	40	3555	1	18	1
5	23	4	10	90	2	3146	29	6	63	19
4	2	1	20	1	3407	4	7	20	15	88
3	2	10	17	3511	1	64	1	13	64	31
2	11	17	3505	71	27	10	12	41	22	11
1	0	4070	19	16	15	15	4	33	34	16
0	3570	0	16	6	13	17	26	5	13	19

Statistics	values
Accuracy	0.9496
95% CI	0.9473, 0.9518

The model performed exceedingly well when compared to previous models achieving an overall accuracy rate of about 95%. This underlines how changing the kernel function in SVM can significantly impact model performance.

4-Conclusion:

In this investigation of the MNIST dataset through various machine learning techniques, we aimed to develop models that could accurately predict handwritten digits from their pixel representations. Our exploration spanned from basic feature engineering to more sophisticated regularization and support vector machines.

Upon delving into more advanced methodologies, it became evident that both LASSO-regularized multinomial logistic regression and SVM with different kernels significantly outperformed our initial logistic models. Regularization introduced an approach to feature selection, thereby improving prediction performance without overfitting, while SVMs take advanage of kernel tricks to project data into higher dimensions where classes are more easily separable.

The best-performing model emerged from the use of an SVM with a polynomial kernel achieving an accuracy rate of approximately 95%. To statistically validate these observations, we perform hypothesis testing on accuracies obtained from the two top-performing models: LASSO-regularized logistic regression and polynomial-kernel SVM, to ascertain if differences in their performances are indeed significant beyond random chance variations. We so tested the significance with McNemar's test, given that the models are tested on the same set of instances, with null hypothesis that there is no difference between the two models:

```
##
##  McNemar's Chi-squared test with continuity correction
##
## data:  tab
## McNemar's chi-squared = 28218, df = 1, p-value < 2.2e-16
```

Given the McNemar's chi-squared test has a very large statistic and the p-value is therefore very small, we will reject the null-hypothesis of the two models predicting at the same accuracy. We can therefore conclude that the model: polynomial-kernel SVM scores the best in our analysis. However, this model is still imperfect, as it can predict with an accuracy of ~95%. For future research, we recommend to apply different neural networks on this data set to see if the model can further improve.