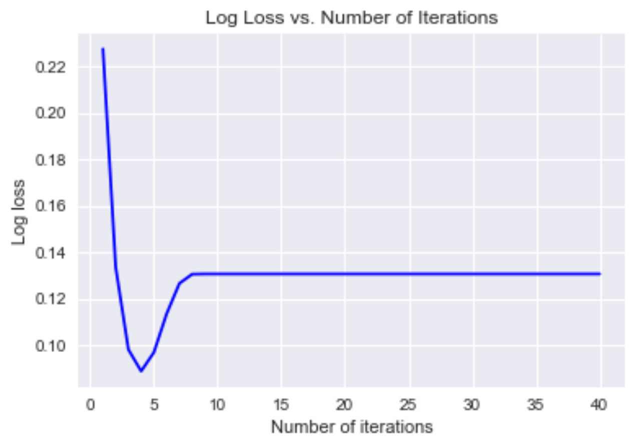
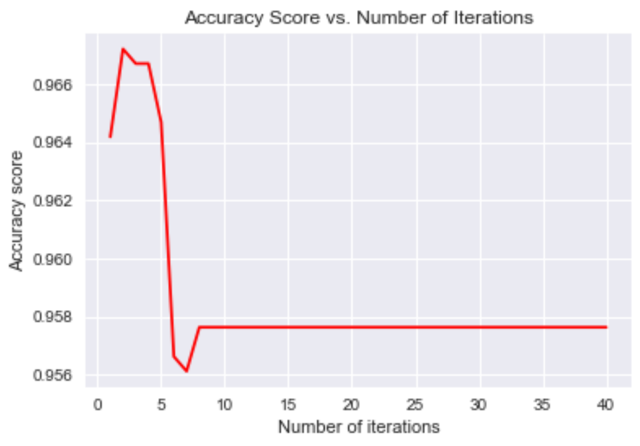
Ercan Sen
COMP 135: Project 1

Digit Classification with Logistic Regression

Tuning Number of Iterations for Convergence Hyperparameter

We tune the hyperparameter max_iter for sklearn's Logistic Regression model by trying out values in the range 1-40.

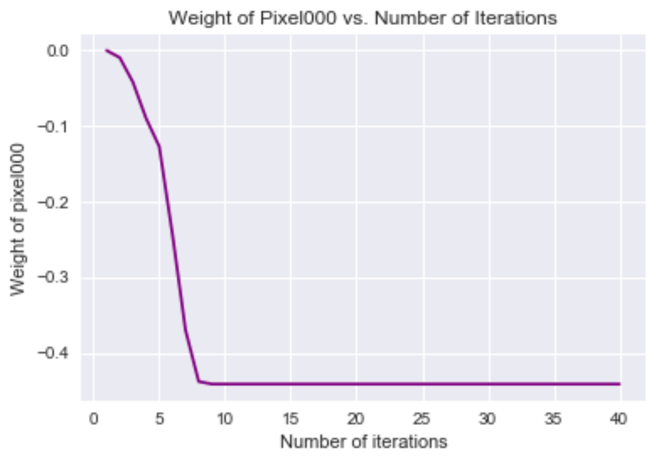




The figure on the left shows log loss, while the one of the right shows accuracy score, both as a function of varying values of max_iter in our 40 trials. The hyperparameter max_iter is directly related to model's convergence with the given datapoints. After enough number of iterations is allowed, the model will converge and become stable, hence perform the best. By definition, convergence would imply a 'reaching a plateau' kind of behavior; that plateau would be when the model is most stable. It is only then that it would be useful to apply the model to the task.

If only the log loss and accuracy score metrics for these 40 trials are examined, one could claim that max_iter = 4 is the optimal value for the hyperparameter, since log loss is lowest and accuracy score is highest. However, it is deceiving to think so, since at that point, model has not converged yet, hence is unstable and the metrics do not produce reliable results. The convergence warnings we get for the first 9 values of max_iter supports this claim. The 'plateau' that both plots reach after max_iter = 10 is what we desire to see; that is the point where the model converges, and hence is stable. The next plot will also support our claim.

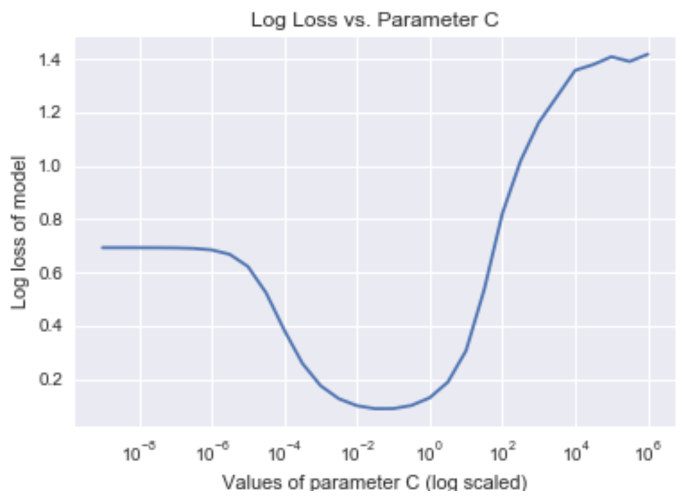
Convergence of Weight of Pixel000



This figure shows the weight of pixel000 (the pixel at the upper left corner of the input images) as a function of the 40 different models we have produced with differing values for max_iter hyperparameter. Since this is a digit classification task, the background, which constitutes the majority of pixel features, do not have an importance. The upper left pixel has no value in identifying the digit, since nobody (usually) starts writing a digit from that part of the allocated space. The weight of the pixel should also reflect this situation, and it is only possible if it converges to its final value (which is the value that is most suitable to the intuitive explanation we have just done). The plot shows that the weight of pixel000 keeps being updated by the model up until a value of 10 for max_iter hyperparameter, and then reaches a constant. This confirms our former claim that a lower hyperparameter value would result in an unstable model. Before 10 iterations, the model has not learned the inputs well enough to make a final decision on what the weights for each feature should be. Yet, 10 iterations is sufficient for it to converge and hence can be used for our classification task.

Tuning C Hyperparameter

To tune the C hyperparameter for Logistic Regression, we create a logspace of 31 values, called C_grid. For each value in C_grid, we build a Logistic Regression model with that value for C, and evaluate model performance by measuring log loss. Here is a plot of log loss values as a function of various values of C (log-scaled, since it was a logspace):



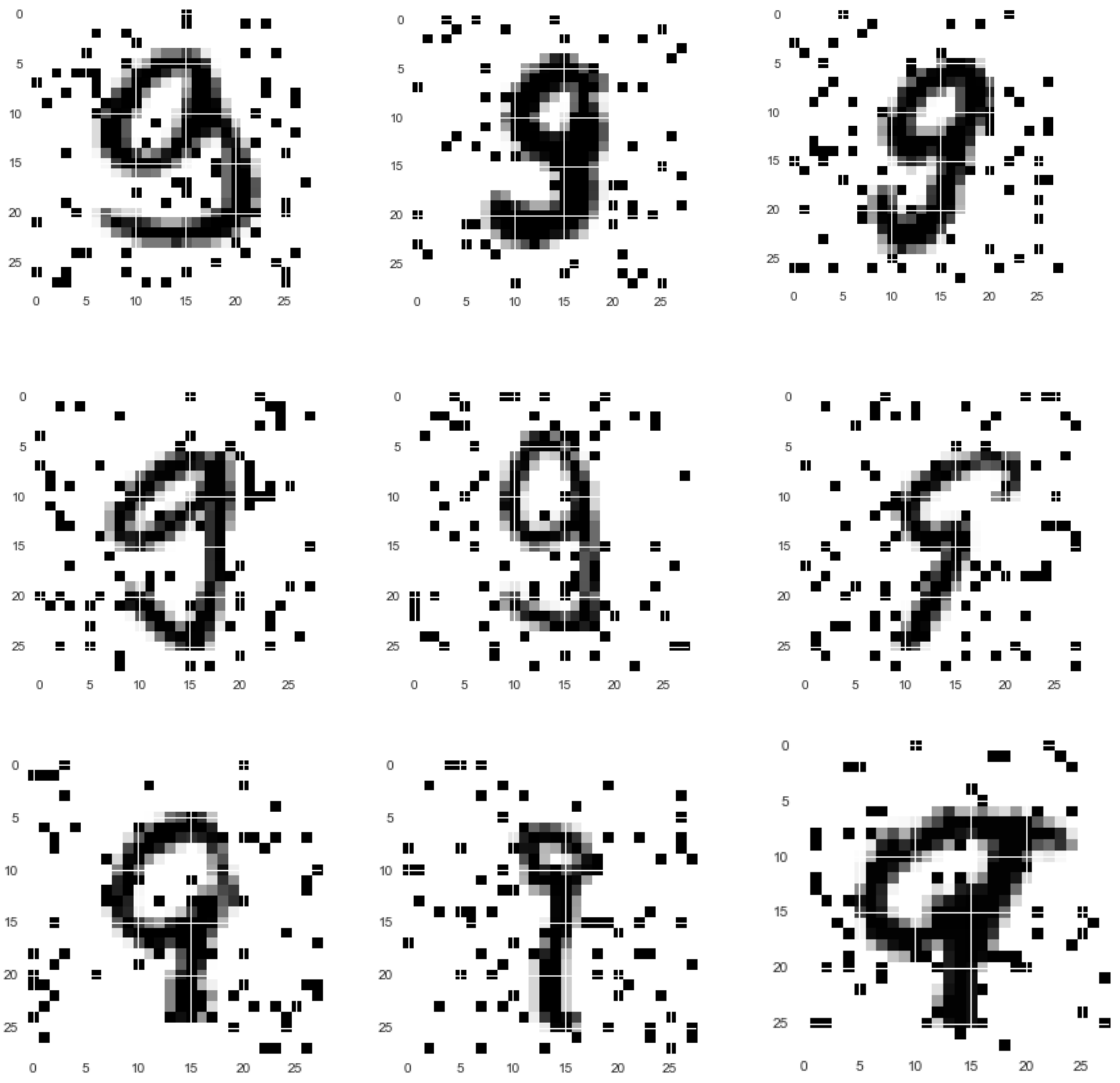
The plot shows that log loss is minimized for a value of C that is between 0.01 and 1. By examining our findings, we find out that the optimal value for C is ***0.0316*** (in 3 significant figures). The corresponding Logistic Regression model has an accuracy score of ***0.967*** (in 3 s.d.) on the test set, which shows that the model performs very well in the classification task. Here is the confusion matrix for the model:

	Predicted 8	**Predicted 9**
Actual 8	942	32
Actual 9	33	976

Considering 8 refers to the negative (0) class, and 9 to the positive (1), there are 32 False Negative classifications and 33 False Positive classifications made by the optimal model. We will examine what might have caused these false classifications by looking at sample FN and FP-classified digits.

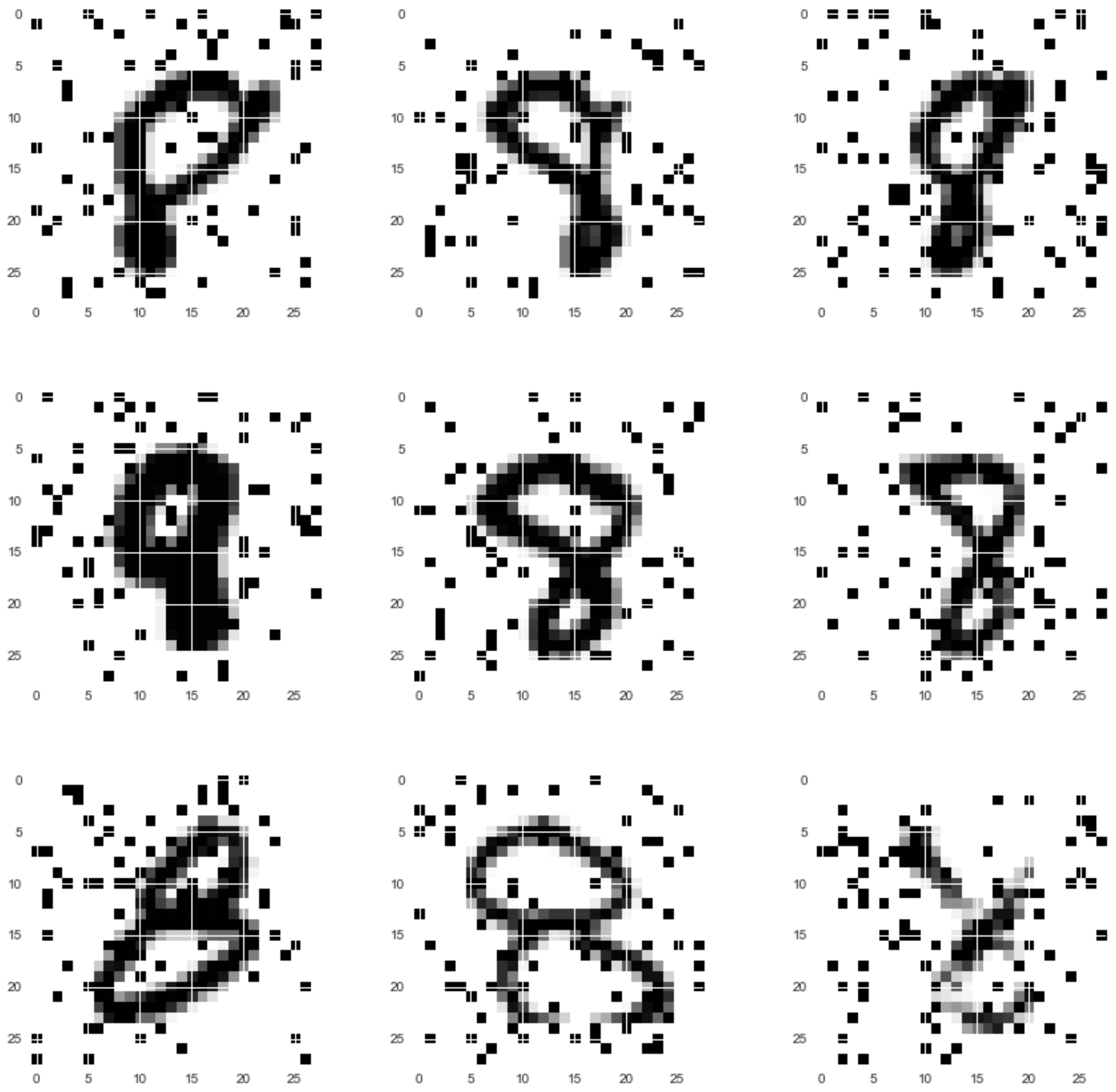
Sample FN and FP Images

FN Classifications:



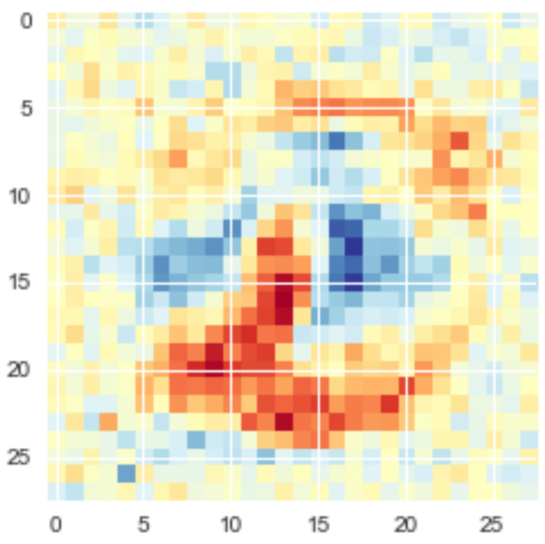
These 9 images are randomly sampled from the total number of 32 FN classifications the best model has made. The common thing these all share are that they all exhibit a handwritten digit 9 (positive class) that is not written neatly/conventionally. It would be hard to classify the image on row 2, column 3 (shortly (2,3) hereafter) correctly even with our own eyes. The digits on (1,1), (1,2), (1,3), (2,1) & (2,2) all have an unconventionally long and curvy tail, easily confusable with the bottom circle of digit 8 especially given the amount of noise. Bottom row images all are disproportional visual representations of the digit 9: (3,1) & (3,3) have very large upper circles compared to their tails, while (3,2) has a small circle and a relatively long tail.

FP Classifications:



These 9 images are randomly sampled from the 33 FP classifications the best model has made. A common thing these all exhibit is how asymmetric they are. It would not be realistic to expect a handwritten digit 8 to as symmetric as a digital or a typewritten one; yet these examples seem to be forcing the limits, and hence were falsely classified to be in the positive class (9). Images on the first row and (2,1) have such a small and skewed bottom circles that they look like a block/line, almost as if it is the tail of 9. Images (2,2), (2,3) & (3,1), although I suspect that without the noise could have been classified correctly, have either one of the top or the bottom circle relatively larger than the other. The last 2 images do not even bother to complete the full edges of digit 8, as if the pen dried out in the middle of the process.

Weights of Classifier



This heat map shows the weight value the classifier with the most accurate predictions (optimal C score) assigned to each pixel feature after converging on and evaluating the model. It is a map of the colors red, yellow and blue. The color yellow, as well as the paler shades of red, orange and blue, represent that the relation to either classification is equally distant. In other words, the pixels at those specific regions of the image do not give us much value and power to classify the image into either category (pos.=9, neg.=8). The dark red curve, like a slightly bent letter C, starting around position [12,12] of the matrix, going as far to the left as [20,7], and curving rightwards until [23,20]. That is showing us a strong relationship. It is visually similar to the left half of the lower circle of 8. There is also a strong, dark blue relationship around the middle region of the image. It is like a short, vertical line between [12,17] and [16,17]. It seems to be related to the right end of the digit 9, like where the circle and tail of the digit merges, as opposed to the portion of the digit 8 where the top and bottom circles merge that is on the middle portion of the image (part of the dark red, orange curve that was explained previously). To conclude, as the pixels at the respective visual fields show, the red features support the classification as 8, while the blue features support the classification as 9.