Homework 6

Han Yong Wunrow
Student No: 1877282
AMATH 584

Due: Monday, December 14, 2020

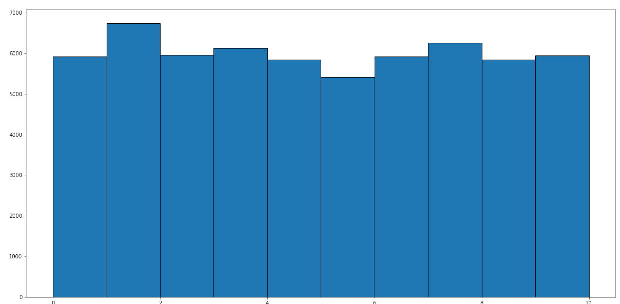
Abstract

In this report, we build a linear classifier on the MNIST database of handwritten digits. We classify an image as either a 1,2,3,4,5,6,7,8,9, or 0 using logistic regression with both LASSO (using the ℓ_1-norm), Ridge (using the ℓ_2-norm), and Elastic Net (using both the ℓ_1- and ℓ_2 norm). We use Grid Search to tune the hyperparameters λ_1, λ_2, and `l1_ratio` using k-fold crossvalidation with $k = 3$. We rank each pixel using the absolute value of the coefficients for each of the models and subset the training and test sets to the top 100 pixels. When evaluating on the full test set, LASSO had the highest test accuracy with 92.62%, followed by Ridge with 92.54% and Elastic Net with 92.52%. However, after subseting to the top 100 pixels, Elastic Net had the highest accuracy of 87.5% followed by Ridge with 88.3% and LASSO with 79.7%. We used Python 3 and Google Collaboratory for this analysis, and code is provided on GitHub.

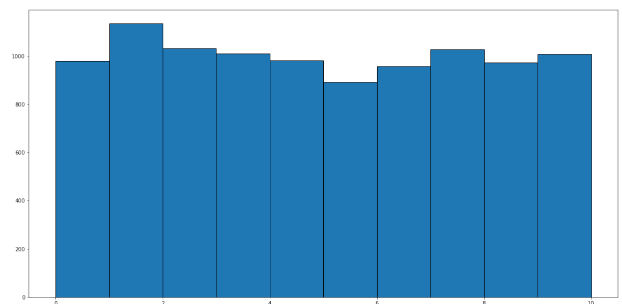
The MNIST database contains 60,000 images of handwritten digits (0 through 9) that are all labeled in a training set and 10,000 labeled images in a test set. Each image is 28 by 28 pixels.

We reshape each image in our training set as a 784×1 column vector x_i and load into a matrix A. So each column in A represents an image, and A has 784 rows and 60,000 columns. We perform a one-hot encoding of the labels so that $y_i = e_k$ (the standard basis vector) corresponds to image i having been labeled as the integer $(k-1)$ for $0 \leq k \leq 9$. We load each column vector y_i into a matrix B. So B has 10 rows and 60,000 columns. So $A^T X = B^T$ is an overdetermined system where X has 784 rows and 10 columns. Since we are inclusing an intercept in our model, we add an additional column $(1, 1, \ldots, 1)^T$ to A^T and X. So our feature space are the 784 pixels.

Both the training and test sets are uniformly distributed with similar counts for each digit so we do not have to worry about balancing. Also these data sets are generalizable to real-world scenarios where each digit is equally likely to appear.



(a) Training set histogram for each digit



(b) Test set histogram for each digit

Each image has been centered and the pixel value ranges from 0 to 255. We normalize the training and test sets by dividing by the max value 255.

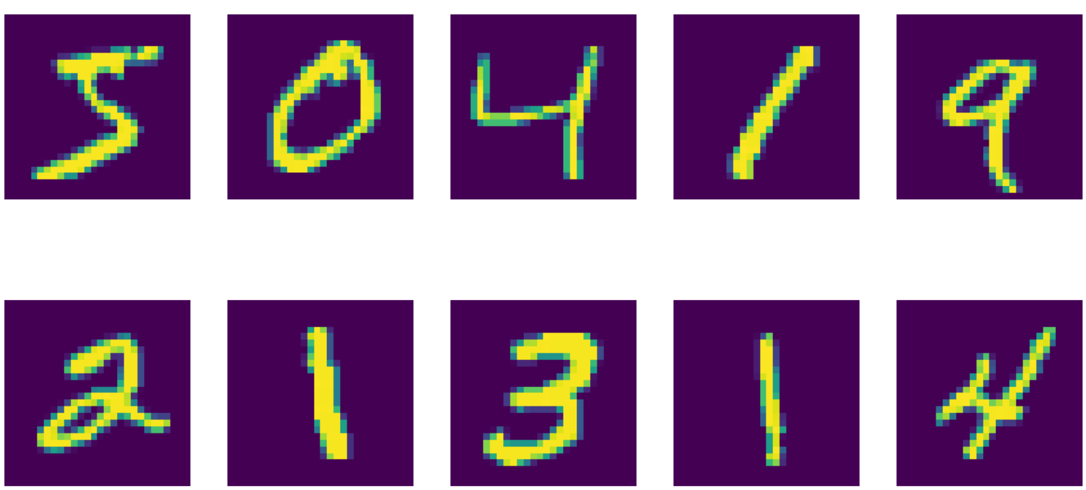


Figure 2: First 10 digits in trainning set

2 Logistic Regression with LASSO, Ridge, and Elastic Net

We classify each image in our training set as either a 1,2,3,4,5,6,7,8,9, or 0 using logistic regression with LASSO (using the ℓ_1-norm), Ridge (using the ℓ_2-norm), and Elastic Net (using both the ℓ_1- and ℓ_2 norm). The respective optimization problems for LASSO, Ridge, and Elastic Net are

$$\max_{\beta} \sum_{i=1}^{n} \sum_{k=1}^{9} \mathbb{1}_{y_i=k} \log P(y_i = k \mid x_i, \beta) - \lambda_1 \|\beta\|_1 \tag{1}$$

$$\max_{\beta} \sum_{i=1}^{n} \sum_{k=1}^{9} \mathbb{1}_{y_i=k} \log P(y_i = k \mid x_i, \beta) - \lambda_1 \|\beta\|_2^2 \tag{2}$$

$$\max_{\beta} \sum_{i=1}^{n} \sum_{k=1}^{9} \mathbb{1}_{y_i=k} \log P(y_i = k \mid x_i, \beta) - \lambda_1 \|\beta\|_1 - \lambda_2 \|\beta\|_2^2 \tag{3}$$

where the conditional probabilities are computed using softmax

$$P(y = k \mid x, \beta) = \frac{\exp(\beta_k^T x)}{\sum_{j=1}^{9} \exp(\beta_i^T x)}.$$

Our coefficients β are used the predict the label $\hat{y}$ for a given image x via

$$\hat{y} = \text{argmax}_k \frac{\exp(\hat{\beta}_k^T x)}{\sum_{j=1}^{9} \exp(\hat{\beta}_i^T x)}$$

Equations (1),(2),and(3) are rewritten using scikit-learn's objectives

$$\max_{\beta} \sum_{i=1}^{n} \sum_{k=1}^{9} \mathbb{1}_{y_i=k} \log P(y_i = k \mid x_i, \beta) - \frac{1}{C} \|\beta\|_1 \tag{4}$$

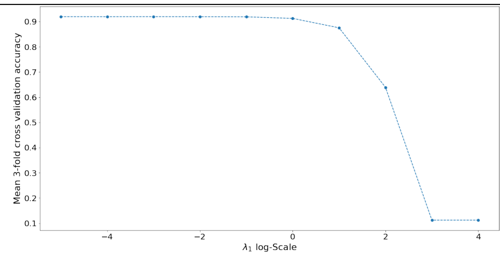
where $\lambda_1 = 1/C$,

$$\max_{\beta} \sum_{i=1}^{n} \sum_{k=1}^{9} \mathbb{1}_{y_i=k} \log P(y_i = k \mid x_i, \beta) - \frac{1}{C} \|\beta\|_2^2 \tag{5}$$

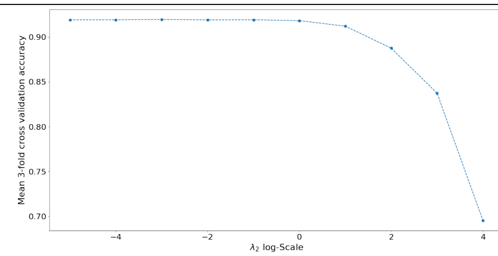
where $\lambda_2 = 1/C$, and

$$\max_{\beta} \sum_{i=1}^{n} \sum_{k=1}^{9} \mathbb{1}_{y_i=k} \log P(y_i = k \mid x_i, \beta) - \frac{\texttt{l1\_ratio}}{C} \|\beta\|_1 - \frac{1 - \texttt{l1\_ratio}}{C} \|\beta\|_2^2 \tag{6}$$

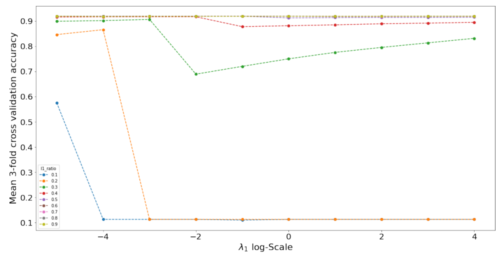
where $\lambda_1 = \texttt{l1\_ratio}/C$ and $\lambda_2 = (1 - \texttt{l1\_ratio})/C$. We use Grid Search to tune the hyperparameters C and `l1_ratio` using k-fold crossvalidation with $k = 3$. The parameter space that we searched was $C \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2 10^3, 10^4\}$ and $\texttt{l1\_ratio} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. We plot the 3-fold mean validation accuracy in Figure 3. Elastic Net had the highest validation accuracy of 91.97%, followed by LASSO and Ridge both with 91.95% accuracy. The optimal parameters for each model are provided in the caption of Figure 3.



(a) LASSO validation accuracy



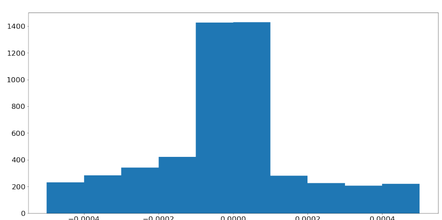
(b) Ridge validation accuracy



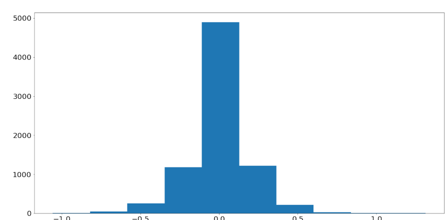
(c) Elastic Net validation accuracy

Figure 3: The parameters with the highest 3-fold mean validation accuracy was (a) $\lambda_1 = 10^{-2}$ for LASSO, (b) $\lambda_2 = 10^{-4}$ for Ridge, and (c) $\lambda_1 = 10^{-2}$, `l1_ratio` $= 0.8$ for Elastic Net.

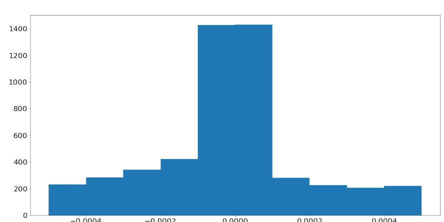
Since the ℓ_1-norm induces sparsity, LASSO had the highest percent of features with coefficient 0 (18.62%) followed by Elastic Net with 15.22% and Ridge with 8.55% as seen in Figure 4. LASSO has advantages over Ridge regression since we can reduce the overdetermined system $A^T X = B^T$ to have fewer columns with LASSO where we can remove the pixels that have a corresponding coefficient of zero.



(a) LASSO



(b) Ridge



(c) Elastic Net

Figure 4: (a) 18.62% of features are 0 for LASSO, (b) 8.55% of features are 0 for Ridge, and (c) 15.22% of features are 0 for Elastic Net

We visualize the coefficient values in Figures 5, 6, and 7. We can see the where the digits are written in blue and the outline of the digit in red. As expected, the white pixels near the boarder to not contribute information towards categorizing the image.

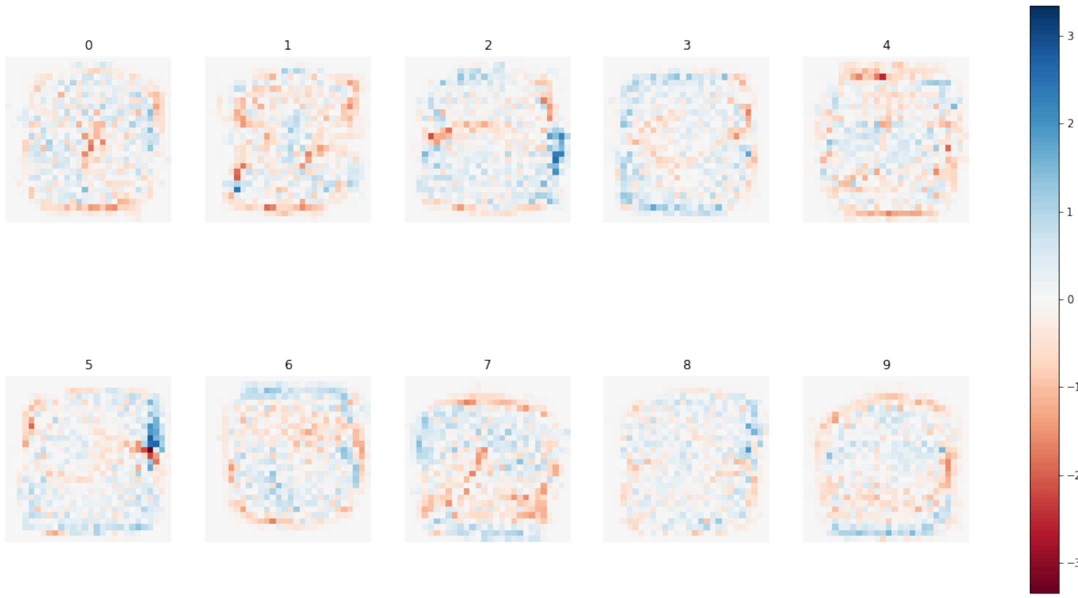


Figure 5: Coefficients for LASSO model

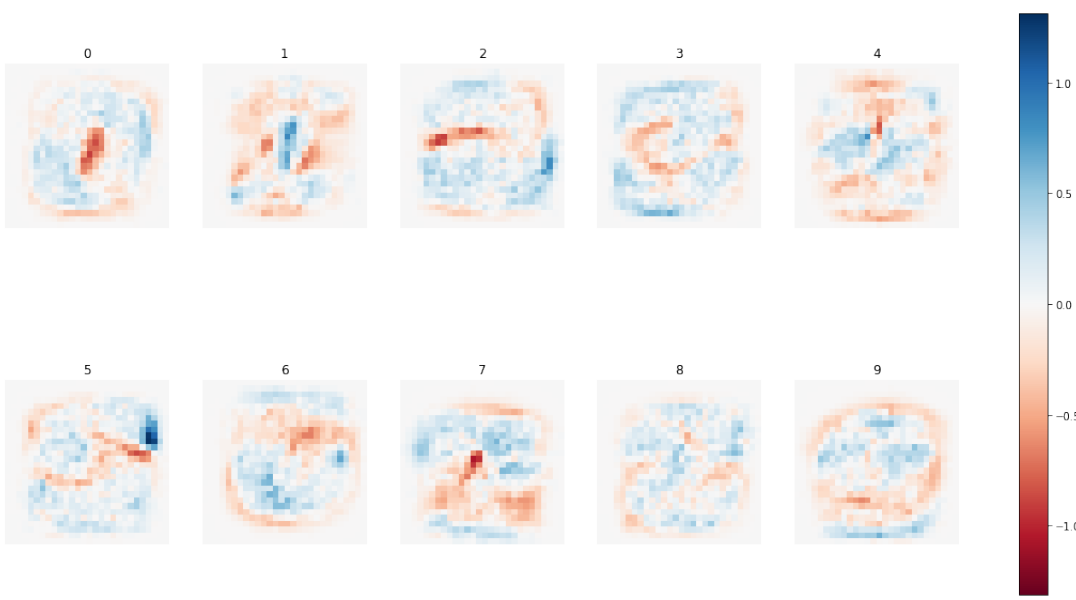


Figure 6: Coefficients for Ridge model

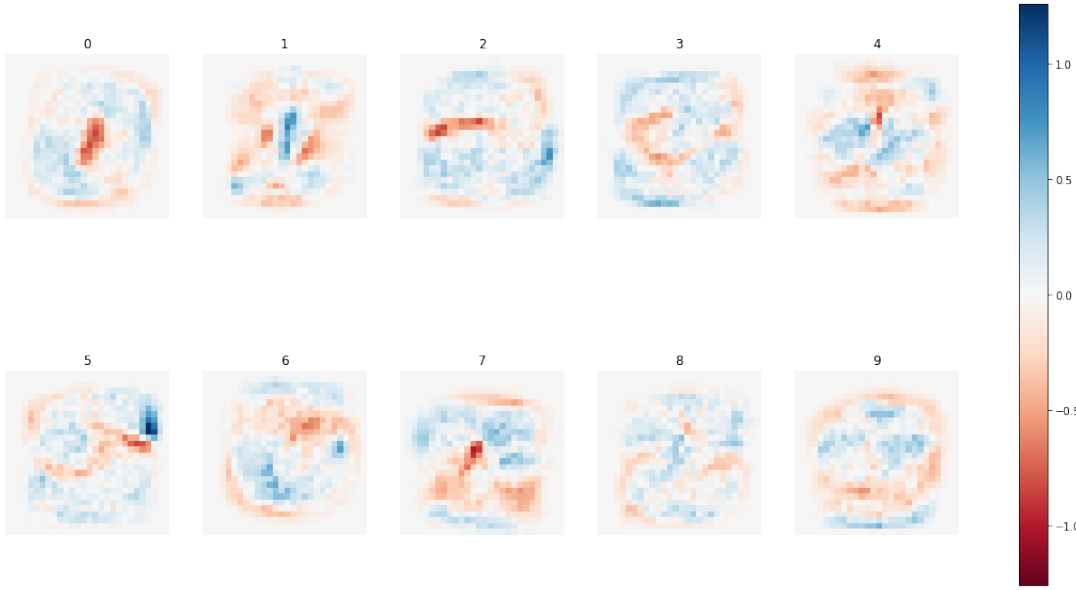


Figure 7: Coefficients for Elastic Net model

3 Ranking Pixels

For each model, we use the absolute value of the coefficient $|\beta_{ij}|$ to rank each pixel j for digit i. In figures 8, 9, and 10 we plot the top 100 coefficients for each model.

To rank each pixel across all digits, we take the sum

$$\sum_{i=0}^{9} |\beta_{ij}| \tag{7}$$

to rank pixel j for $1 \leq j \leq 784$. Again we plot the top 100 pixels for each method in Figures 11. Ridge and Elastic Net seem to rank pixels on the inside border higher, while LASSO ranks pixels on the outside boarder higher.

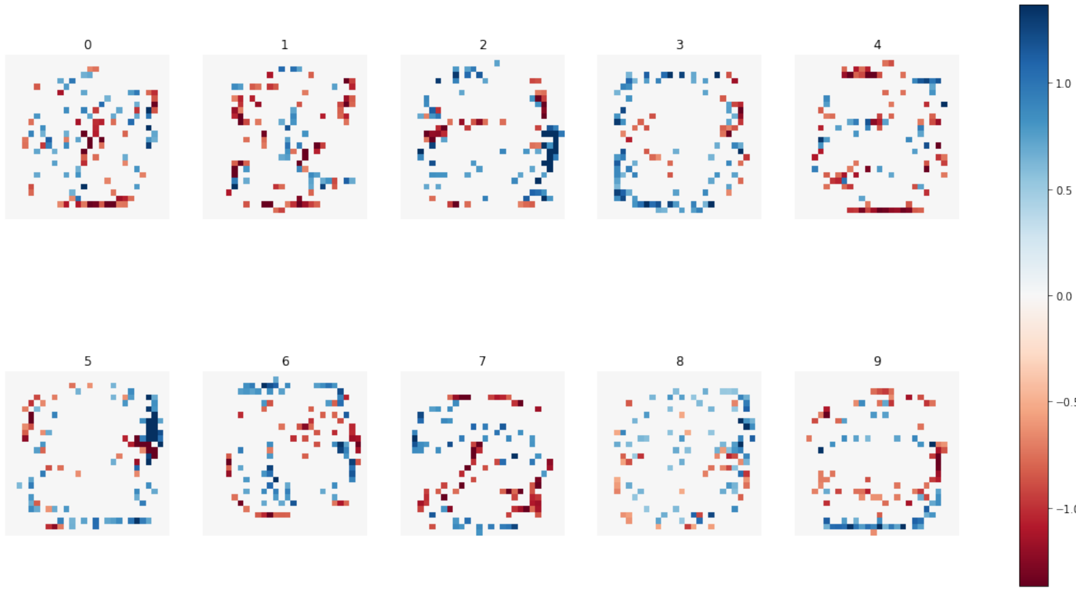


Figure 8: Each digit top 100 coefficients for LASSO model

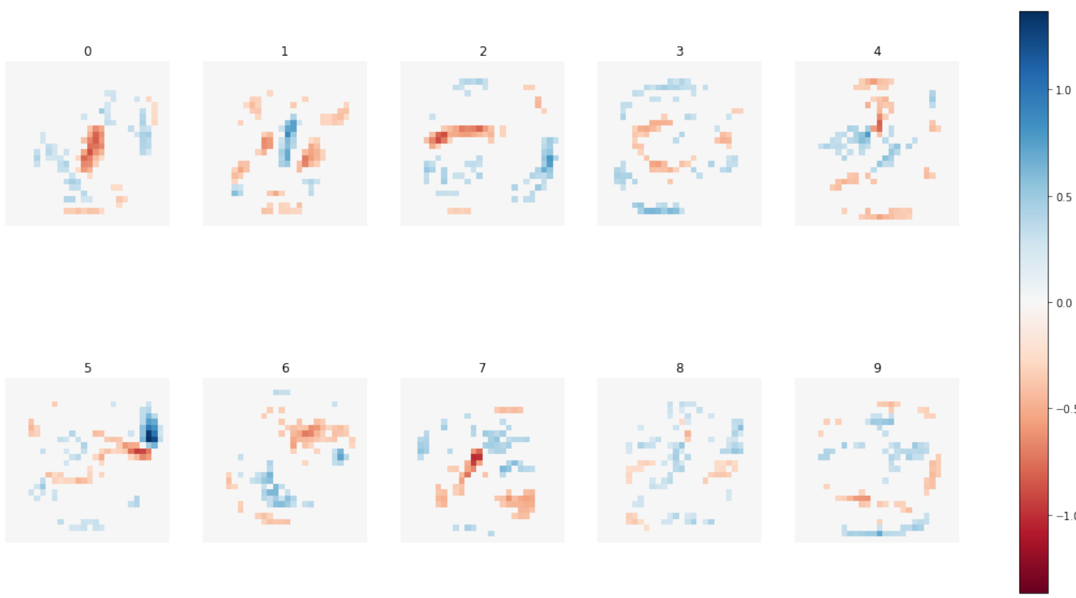


Figure 9: Each digit top 100 coefficients for Ridge model

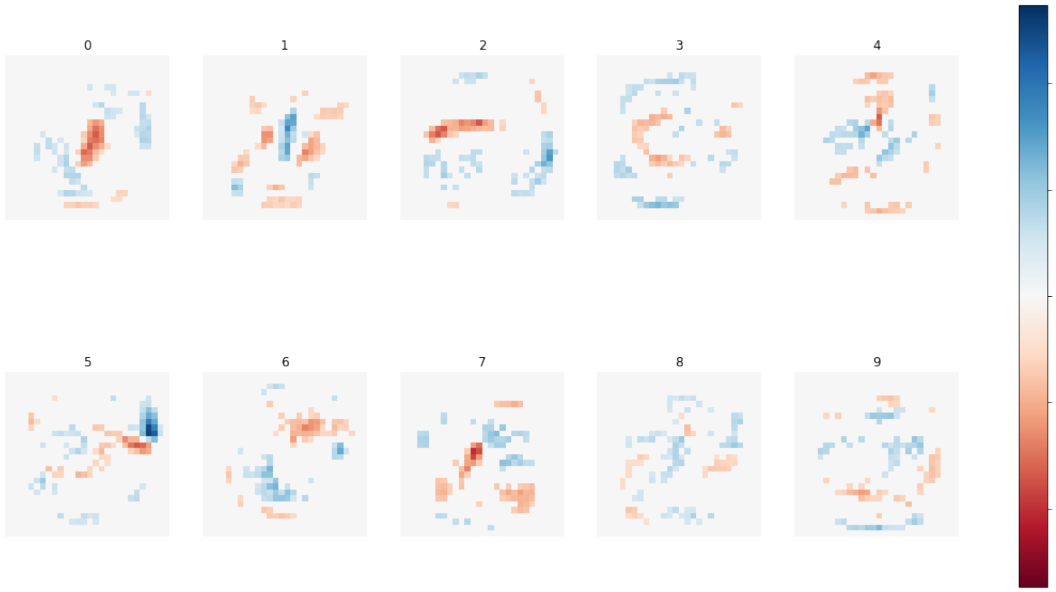
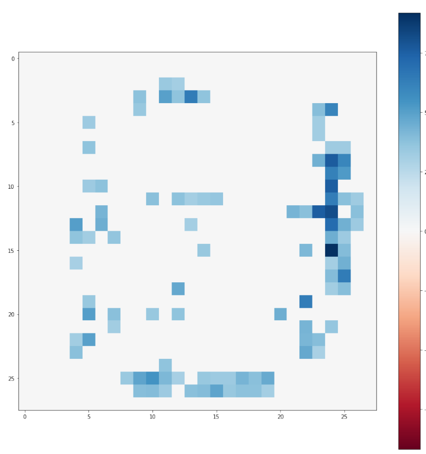
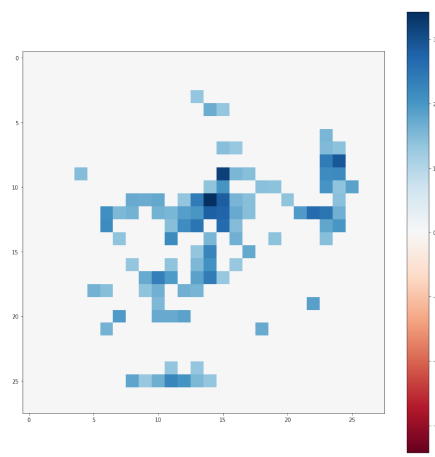


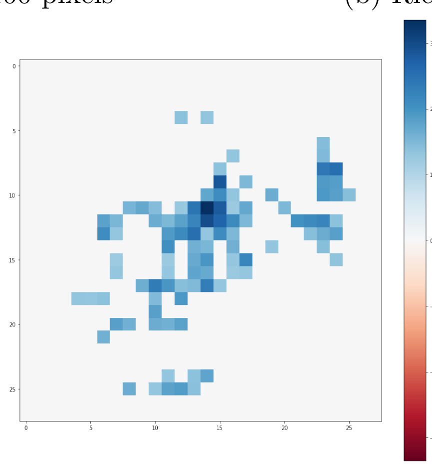
Figure 10: Each digit top 100 coefficients for Elastic Net model



(a) LASSO Top 100 pixels



(b) Ridge Top 100 pixels



(c) Elastic Net Top 100 pixels

Figure 11: Top 100 pixels for each method using equation (7) to rank

4 Test Set Evaluation

First, we evaluate the full models on the test set of 10,000 images. LASSO had the highest test accuracy with 92.62%, followed by Ridge with 92.54% and Elastic Net with 92.52%. We also compute the precision, recall, and F_1 Scores for each digit where

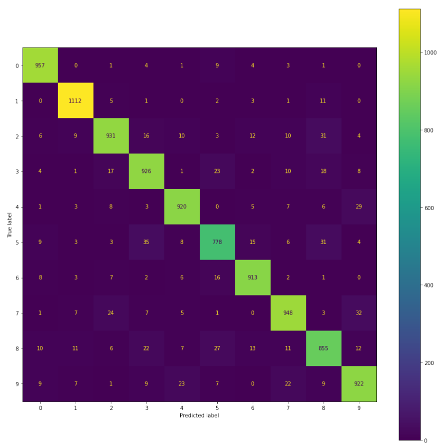
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

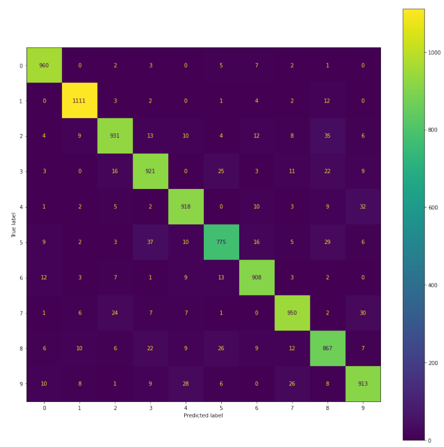
and

$$F_1 = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

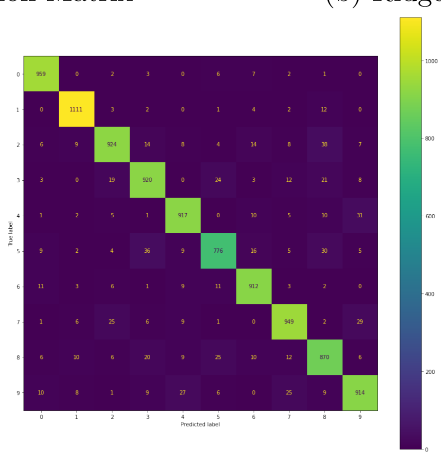
TP stands for true positive, FP stands for false positive, and FN stands for false negative. LASSO had a higher F_1 score for 4s, 6s, and 9s than Ridge and Elastic Net.



(a) LASSO Confusion Matrix



(b) Ridge Confusion Matrix



(c) Elastic Net Confusion Matrix

Figure 12: Evaluated on full test set

	precision	recall	f1-score
0	0.952	0.977	0.964
1	0.962	0.980	0.971
2	0.928	0.902	0.915
3	0.903	0.917	0.910
4	0.938	0.937	0.937
5	0.898	0.872	0.885
6	0.944	0.953	0.949
7	0.929	0.922	0.926
8	0.885	0.878	0.881
9	0.912	0.914	0.913

Table 1: LASSO

	precision	recall	f1-score
0	0.954	0.980	0.967
1	0.965	0.979	0.972
2	0.933	0.902	0.917
3	0.906	0.912	0.909
4	0.926	0.935	0.931
5	0.905	0.869	0.887
6	0.937	0.948	0.942
7	0.930	0.924	0.927
8	0.878	0.890	0.884
9	0.910	0.905	0.908

Table 2: Ridge

	precision	recall	f1-score
0	0.953	0.979	0.966
1	0.965	0.979	0.972
2	0.929	0.895	0.912
3	0.909	0.911	0.910
4	0.928	0.934	0.931
5	0.909	0.870	0.889
6	0.934	0.952	0.943
7	0.928	0.923	0.925
8	0.874	0.893	0.884
9	0.914	0.906	0.910

Table 3: Elastic Net

Table 4: Classification metrics for full models

4.1 Top 100 pixel Test Set Evaluation

We analyze how our models perform on the top 100 pixels as show in Figure 11. We subset both the training and test sets to the top 100 models and train three models using LASSO, Ridge, and Elastic Net regularization using the parameters as stated in the caption of Figure 3. We evaluated our models on the sparse test sets and compute the confusion matrix, accuracy, precision, recall, and F_1 scores. In this setting, Elastic Net has the highest accuracy of 87.5% followed by Ridge with 88.3% and LASSO with 79.7%. Ridge had a higher F_1 score than LASSO and Elastic Net for all digits.

Subsetting our images to these top 100 pixels decreases the total number of pixels 87.2%, which when scaled will decrease storage requirements and runtime. However, with this reduction we saw a reduction of approximately 5% in our test accuracy for Ridge Regression.

	precision	recall	f1-score
0	0.882	0.926	0.903
1	0.862	0.949	0.904
2	0.871	0.829	0.850
3	0.733	0.778	0.755
4	0.705	0.756	0.730
5	0.745	0.629	0.682
6	0.800	0.841	0.820
7	0.840	0.841	0.841
8	0.748	0.630	0.684
9	0.754	0.749	0.751

Table 5: LASSO

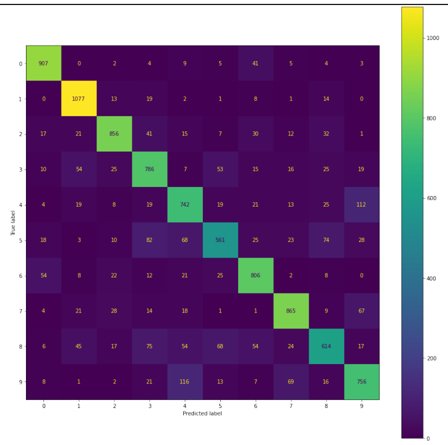
	precision	recall	f1-score
0	0.929	0.974	0.951
1	0.937	0.963	0.950
2	0.910	0.869	0.889
3	0.865	0.879	0.872
4	0.856	0.853	0.855
5	0.865	0.825	0.845
6	0.878	0.898	0.888
7	0.892	0.898	0.895
8	0.864	0.811	0.837
9	0.824	0.843	0.833

Table 6: Ridge

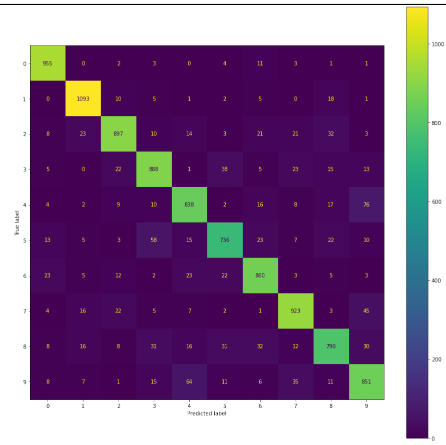
	precision	recall	f1-score
0	0.932	0.971	0.951
1	0.927	0.969	0.947
2	0.896	0.840	0.867
3	0.875	0.857	0.866
4	0.836	0.838	0.837
5	0.853	0.835	0.844
6	0.885	0.887	0.886
7	0.880	0.894	0.887
8	0.849	0.813	0.831
9	0.802	0.825	0.813

Table 7: Elastic Net

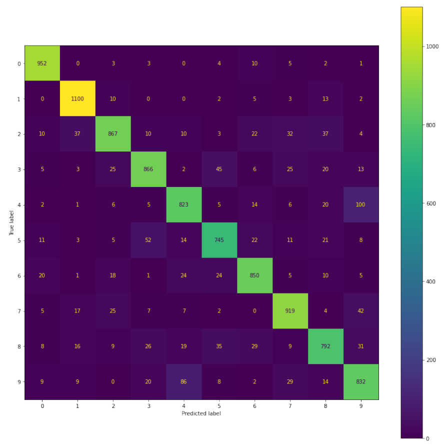
Table 8: Classification metrics for sparse top 100 pixel models



(a) LASSO Confusion Matrix



(b) Ridge Confusion Matrix



(c) Elastic Net Confusion Matrix

Figure 13: Evaluated on sparse top 100 pixel test set

5 Appendix Code

All analysis was completed using Python 3 in a Google Collaboratory Jupyter Notebook. Libraries used include `google.colab` for file management; `numpy` for linear algebra; `matplotlib` for plotting; `tensorflow` for loading the mnist dataset; and `scikitlearn` for machine learning. Code for this analysis is provided below and on GitHub.

```python
# -*- coding: utf-8 -*-
"""hwunrow_amath584_hw6.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1okPbSGqe5fgb_keR9biC92qE0xrbCd3r
"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import pandas as pd
import tensorflow as tf

import matplotlib.pyplot as plt
# %matplotlib inline

from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.utils import to_categorical

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

from google.colab import drive
drive.mount('/content/drive')

"""# Data Processing and EDA"""

data_dir = "/content/drive/MyDrive/School/AMATH584/repos/amath584/hw6/data/mnist.npz"
(x_train, y_train), (x_test, y_test) = load_data(path=data_dir)

print(f"training set dimensions: {x_train.shape}")
print(f"test set dimensions: {x_test.shape}")

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# one-hot encode
y_train_encode = to_categorical(y_train, 10)
y_test_encode = to_categorical(y_test, 10)

"""Plot the first 10 numbers in training set"""

fig=plt.figure(figsize=(20,10))

rows = 2
columns = 5

for i in range(10):
  ax = fig.add_subplot(rows, columns, i+1)
  plt.imshow(x_train[i].reshape(28,28))
  plt.axis('off')

```

```
x_train.max()

# normalize
x_train = x_train/255
x_test = x_test/255

np.histogram(y_test, bins=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

fig=plt.figure(figsize=(20,10))

plt.hist(y_train, bins=[0,1,2,3,4,5,6,7,8,9, 10], ec='black')

fig=plt.figure(figsize=(20,10))

plt.hist(y_test, bins=[0,1,2,3,4,5,6,7,8,9,10], ec='black')

"""# Logistic Regression with LASSO, Ridge, and Elastic Net

## Hyperparameter Tuning
We use Grid Search to tune the type of regularization (L1 norm or Elastic Net) and size of the penalty.
"""

lr = LogisticRegression(tol=0.1)

param_grid = [
  {'C': [10**x for x in range(-5,5)], 'penalty': ['l1'], 'solver': ['saga']},
  {'C': [10**x for x in range(-5,5)], 'l1_ratio':[0.1*(x+1) for x in range(9)], 'penalty': ['elasticnet'], 'solver': ['saga']},
  ]

clf = GridSearchCV(lr, param_grid, cv=3, scoring='accuracy', n_jobs=-1)

clf.fit(x_train, y_train)

# import pickle
# filename = "/content/drive/MyDrive/School/AMATH584/repos/amath584/hw6/lr_norm_model.sav"
# pickle.dump(clf, open(filename, 'wb'))

# clf = pickle.load(open(filename, 'rb'))

"""### LASSO"""

lasso_accuracy = clf.cv_results_['mean_test_score'][0:10]

lasso_accuracy

plt.figure(figsize=(20,10))
#flip because the hyperparameter in scikit learn C = 1/lambda
plt.plot(list(range(-5,5)), np.flip(lasso_accuracy), '--o')
plt.xlabel(r"$\lambda_1$ log-Scale",fontsize=22)
plt.ylabel("Mean 3-fold cross validation accuracy",fontsize=22)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
```

```python
lasso_best_params = clf.cv_results_['params'][np.argmax(lasso_accuracy)]
clf_lasso = LogisticRegression(**lasso_best_params)

clf_lasso.fit(x_train, y_train)

lasso_best_params

lasso_coefs = clf_lasso.coef_

print(f"{np.mean(lasso_coefs == 0) * 100:.2f} % of features are zero!")

plt.figure(figsize=(20,10))
plt.hist(lasso_coefs.flatten())

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20,10))

scale = np.abs(lasso_coefs).max()

for i, ax in enumerate(axes.flat):
  ax.set_axis_off()
  ax.set_title(f"{i}")
  img = lasso_coefs[i]
  img = img.reshape(28,28)
  im = ax.imshow(img, interpolation='nearest', vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

cbar = fig.colorbar(im, ax=axes.ravel().tolist())

"""### Elastic Net"""

elastic_net_accuracy = clf.cv_results_['mean_test_score'][10:101]
elastic_net_params = clf.cv_results_['params'][10:101]

# rows represent same C, columns are for same l1_ratio
elastic_net_accuracy = elastic_net_accuracy.reshape(9,10)
#flip because the hyperparameter in scikit learn C = 1/lambda
elastic_net_accuracy = np.flip(elastic_net_accuracy, axis=1)

plt.figure(figsize=(20,10))
#flip because the hyperparameter in scikit learn C = 1/lambda
plt.plot(list(range(-5,5)), elastic_net_accuracy.T, '--o')
plt.xlabel(r"$\lambda_1$ log-Scale", fontsize=22)
plt.ylabel("Mean 3-fold cross validation accuracy", fontsize=22)

plt.legend([round(0.1*(x+1),1) for x in range(9)], title="l1_ratio")

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

clf.best_params_

coefs = clf.best_estimator_.coef_

print(f"{np.mean(coefs == 0) * 100:.2f} % of features are zero!")
```

```python

plt.figure(figsize=(20,10))
plt.hist(coefs.flatten())

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20,10))

scale = np.abs(coefs).max()

for i, ax in enumerate(axes.flat):
  ax.set_axis_off()
  ax.set_title(f"{i}")
  img = coefs[i]
  img = img.reshape(28,28)
  im = ax.imshow(img, interpolation='nearest', vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

cbar = fig.colorbar(im, ax=axes.ravel().tolist())

"""### Ridge
For comparison, we also fit a logistic regression model with $\ell_2$-norm regularization.
"""

lr = LogisticRegression(tol=0.1)

param_grid = [
  {'C': [10**x for x in range(-5,5)], 'penalty': ['l2'], 'solver': ['saga']},
  ]

clf_ridge = GridSearchCV(lr, param_grid, cv=3, scoring='accuracy', n_jobs=2)

clf_ridge.fit(x_train, y_train)

clf_ridge.best_params_

print(f"LASSO: {lasso_accuracy.max() * 100}")
print(f"Ridge: {clf_ridge.best_score_ * 100}")
print(f"Elastic Net: {clf.best_score_ * 100}")

ridge_accuracy = clf_ridge.cv_results_['mean_test_score']

plt.figure(figsize=(20,10))
#flip because the hyperparameter in scikit learn C = 1/lambda
plt.plot(list(range(-5,5)), np.flip(ridge_accuracy), '--o')
plt.xlabel(r"$\lambda_2$ log-Scale",fontsize=22)
plt.ylabel("Mean 3-fold cross validation accuracy",fontsize=22)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

ridge_coefs = clf_ridge.best_estimator_.coef_

print(f"{np.mean(ridge_coefs == 0) * 100:.2f} % of features are zero!")

```

```
plt.figure(figsize=(20,10))
plt.hist(ridge_coefs.flatten())

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20,10))

scale = np.abs(ridge_coefs).max()

for i, ax in enumerate(axes.flat):
  ax.set_axis_off()
  ax.set_title(f"{i}")
  img = ridge_coefs[i]
  img = img.reshape(28,28)
  im = ax.imshow(img, interpolation='nearest', vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

cbar = fig.colorbar(im, ax=axes.ravel().tolist())

"""# Ranking Pixels
We rank the pixels by each its corresponding coefficient for each digit.

### LASSO
"""

# most important pixels for each digit
lasso_coefs.argmax(axis=1)

n = 100

lasso_coefs_copy = lasso_coefs.copy()

arr = np.empty((0,784), float)
for row in lasso_coefs_copy:
  abs_row = abs(row)
  abs_row.sort()
  thresh = abs_row[-n]
  filter = abs(row) < thresh
  row[filter] = 0
  arr = np.append(arr, np.array([row]), axis=0)


fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20,10))

scale = np.abs(ridge_coefs).max()

for i, ax in enumerate(axes.flat):
  ax.set_axis_off()
  ax.set_title(f"{i}")
  img = arr[i]
  img = img.reshape(28,28)
  im = ax.imshow(img, interpolation='nearest', vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

cbar = fig.colorbar(im, ax=axes.ravel().tolist())
```

```
lasso_all_digit_rank = abs(lasso_coefs_copy).sum(axis=0)

lasso_all_digit_copy = lasso_all_digit_rank.copy()
lasso_all_digit_copy.sort()
thresh = lasso_all_digit_copy[-n]
filter = lasso_all_digit_rank < thresh
lasso_all_digit_rank[filter] = 0

scale = np.abs(lasso_all_digit_rank).max()

fig, ax = plt.subplots(figsize=(15, 15), ncols=1)

im = ax.imshow(lasso_all_digit_rank.reshape(28,28), interpolation='nearest',
               vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

fig.colorbar(im, ax=ax)

"""### Ridge"""

n = 100

ridge_coefs_copy = ridge_coefs.copy()

arr = np.empty((0,784), float)
for row in ridge_coefs_copy:
  abs_row = abs(row)
  abs_row.sort()
  thresh = abs_row[-n]
  filter = abs(row) < thresh
  row[filter] = 0
  arr = np.append(arr, np.array([row]), axis=0)


fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20,10))

scale = np.abs(ridge_coefs).max()

for i, ax in enumerate(axes.flat):
  ax.set_axis_off()
  ax.set_title(f"{i}")
  img = arr[i]
  img = img.reshape(28,28)
  im = ax.imshow(img, interpolation='nearest', vmin=-scale, vmax=scale, cmap=plt.cm.
    RdBu)

cbar = fig.colorbar(im, ax=axes.ravel().tolist())

ridge_all_digit_rank = abs(ridge_coefs_copy).sum(axis=0)

ridge_all_digit_copy = ridge_all_digit_rank.copy()
ridge_all_digit_copy.sort()
thresh = ridge_all_digit_copy[-n]
filter = ridge_all_digit_rank < thresh
ridge_all_digit_rank[filter] = 0

scale = np.abs(ridge_all_digit_rank).max()
```

```
fig, ax = plt.subplots(figsize=(15, 15), ncols=1)

im = ax.imshow(ridge_all_digit_rank.reshape(28,28), interpolation='nearest',
               vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

fig.colorbar(im, ax=ax)

"""## Elastic Net"""

n = 100

en_coefs_copy = clf.best_estimator_.coef_.copy()

arr = np.empty((0,784), float)
for row in en_coefs_copy:
  abs_row = abs(row)
  abs_row.sort()
  thresh = abs_row[-n]
  filter = abs(row) < thresh
  row[filter] = 0
  arr = np.append(arr, np.array([row]), axis=0)


fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20,10))

scale = np.abs(ridge_coefs).max()

for i, ax in enumerate(axes.flat):
  ax.set_axis_off()
  ax.set_title(f"{i}")
  img = arr[i]
  img = img.reshape(28,28)
  im = ax.imshow(img, interpolation='nearest', vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

cbar = fig.colorbar(im, ax=axes.ravel().tolist())

en_all_digit_rank = abs(en_coefs_copy).sum(axis=0)

en_all_digit_copy = en_all_digit_rank.copy()
en_all_digit_copy.sort()
thresh = en_all_digit_copy[-n]
filter = en_all_digit_rank < thresh
en_all_digit_rank[filter] = 0

scale = np.abs(en_all_digit_rank).max()

fig, ax = plt.subplots(figsize=(15, 15), ncols=1)

im = ax.imshow(en_all_digit_rank.reshape(28,28), interpolation='nearest',
               vmin=-scale, vmax=scale, cmap=plt.cm.RdBu)

fig.colorbar(im, ax=ax)

"""# Test set

### LASSO
```

```
"""

yhat =clf_lasso.predict(x_test)

print(f"Test accuracy {accuracy_score(y_test, yhat)*100}%")
confusion_matrix(y_test, yhat)

print(classification_report(y_test, yhat, target_names=['0', '1', '2', '3', '4', '5',
    '6', '7', '8', '9'], digits=3))

fig, ax = plt.subplots(figsize=(15, 15))


plot_confusion_matrix(clf_lasso, x_test, y_test, ax=ax, values_format='.4g')

"""### Ridge"""

confusion_matrix(y_test, clf_ridge.predict(x_test))

print(f"Test accuracy {accuracy_score(y_test, clf_ridge.predict(x_test))*100}%")

print(classification_report(y_test,  clf_ridge.predict(x_test), target_names=['0', '1
    ', '2', '3', '4', '5', '6', '7', '8', '9'], digits=3))

fig, ax = plt.subplots(figsize=(15, 15))

plot_confusion_matrix(clf_ridge, x_test, y_test, ax=ax, values_format = '.4g')

"""### Elastic Net"""

confusion_matrix(y_test, clf.best_estimator_.predict(x_test))

print(f"Test accuracy {accuracy_score(y_test, clf.best_estimator_.predict(x_test))
    *100}%")

print(classification_report(y_test, clf.best_estimator_.predict(x_test), target_names
    =['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], digits=3))

fig, ax = plt.subplots(figsize=(15, 15))

plot_confusion_matrix(clf.best_estimator_, x_test, y_test, ax=ax, values_format = '.4
    g')

"""## Sparse Test Evaluation
Using the top 100 pixels from each model we evaluate on the test set.

### LASSO
"""

lasso_sparse_subset = lasso_all_digit_rank.nonzero()

lasso_sparse_subset

# subset training and test sets to top 100 pixels
x_train_sparse_lasso = np.transpose(np.transpose(x_train)[lasso_sparse_subset])
x_test_sparse_lasso = np.transpose(np.transpose(x_test)[lasso_sparse_subset])

```

```
clf_lasso_sparse = LogisticRegression(C= 100, penalty='l1', solver='saga', tol=0.1)
clf_lasso_sparse.fit(x_train_sparse_lasso, y_train)

yhat =clf_lasso_sparse.predict(x_test_sparse_lasso)
print(f"Test accuracy {accuracy_score(y_test, yhat)*100}%")
print(classification_report(
    y_test, yhat,
    target_names=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'],
    digits=3)
)

fig, ax = plt.subplots(figsize=(15, 15))

plot_confusion_matrix(clf_lasso_sparse, x_test_sparse_lasso, y_test, ax=ax,
    values_format = '.4g')

"""### Ridge"""

ridge_sparse_subset = ridge_all_digit_rank.nonzero()

# subset training and test sets to top 100 pixels
x_train_sparse_ridge = np.transpose(np.transpose(x_train)[ridge_sparse_subset])
x_test_sparse_ridge = np.transpose(np.transpose(x_test)[ridge_sparse_subset])

clf_ridge_sparse = LogisticRegression(C= 10000, penalty='l2', solver='saga', tol=0.1)
clf_ridge_sparse.fit(x_train_sparse_ridge, y_train)

yhat =clf_ridge_sparse.predict(x_test_sparse_ridge)
print(f"Test accuracy {accuracy_score(y_test, yhat)*100}%")
print(classification_report(
    y_test, yhat,
    target_names=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'],
    digits=3)
)

fig, ax = plt.subplots(figsize=(15, 15))

plot_confusion_matrix(clf_ridge_sparse, x_test_sparse_ridge, y_test, ax=ax,
    values_format = '.4g')

"""### Elastic Net"""

en_sparse_subset = en_all_digit_rank.nonzero()

# subset training and test sets to top 100 pixels
x_train_sparse_en = np.transpose(np.transpose(x_train)[en_sparse_subset])
x_test_sparse_en = np.transpose(np.transpose(x_test)[en_sparse_subset])

clf.best_params_

clf_en_sparse = LogisticRegression(C= 100, penalty='elasticnet', l1_ratio=0.8, solver
    ='saga', tol=0.1)
clf_en_sparse.fit(x_train_sparse_en, y_train)

yhat =clf_en_sparse.predict(x_test_sparse_en)
print(f"Test accuracy {accuracy_score(y_test, yhat)*100}%")
print(classification_report(
```

```
    y_test, yhat,
    target_names=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'],
    digits=3)
)

fig, ax = plt.subplots(figsize=(15, 15))

plot_confusion_matrix(clf_en_sparse, x_test_sparse_en, y_test, ax=ax, values_format =
    '.4g')
```