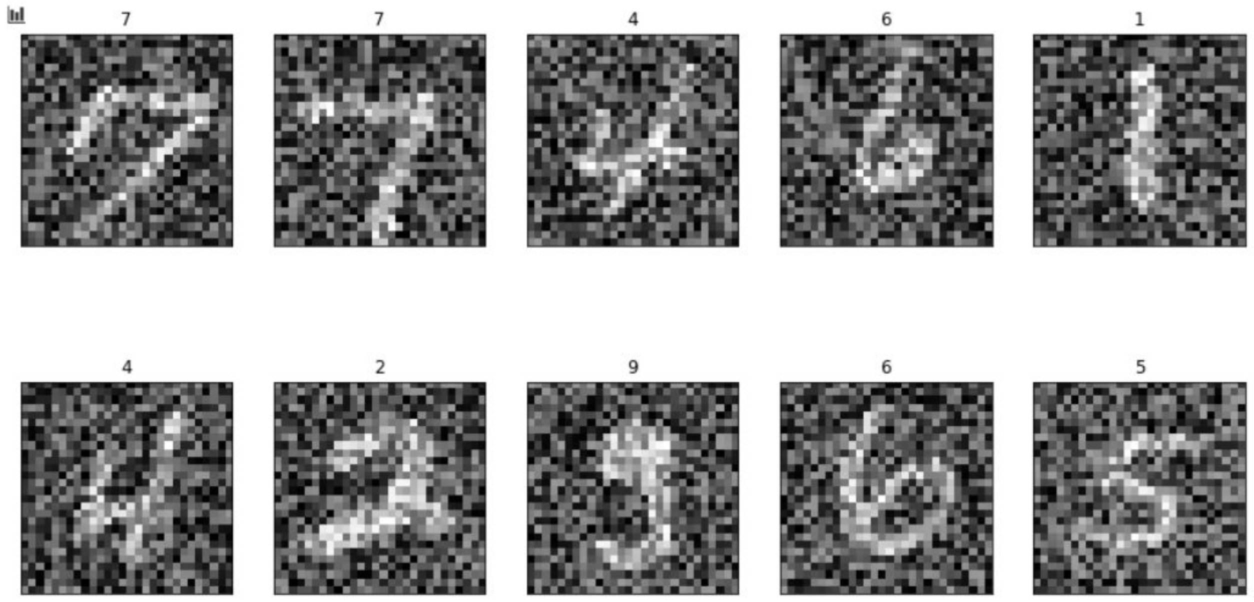
Name: Tai Duc Nguyen - CS 612 - Prof. David Han - 02/28/2021

Project 2 Report

I. Models

In this project, 3 machine learning models are implemented: k-nearest neighbors, support vector machine, and logistic regression to perform digit classification on the MNIST dataset.

Background on the dataset:
The MNIST database (*Modified National Institute of Standards and Technology database*) is a large database of handwritten digits that is commonly used for training various image processing (and computer vision) systems. In order to perform training, we were given 60,000 training images of size 28 by 28 (784) pixels. Through common inspection of these given images, it looks like they are very noisy and the pixel values' amplitudes are not from 0 to 255 (max of 655).



Hence, steps are needed to reduce this noise, purify the images downto an image of only 0 and 1s, just like the original dataset:

We will also provide training results for the raw dataset given to us and prove that reducing the noise gives a much needed boost to the accuracy of the algorithms.

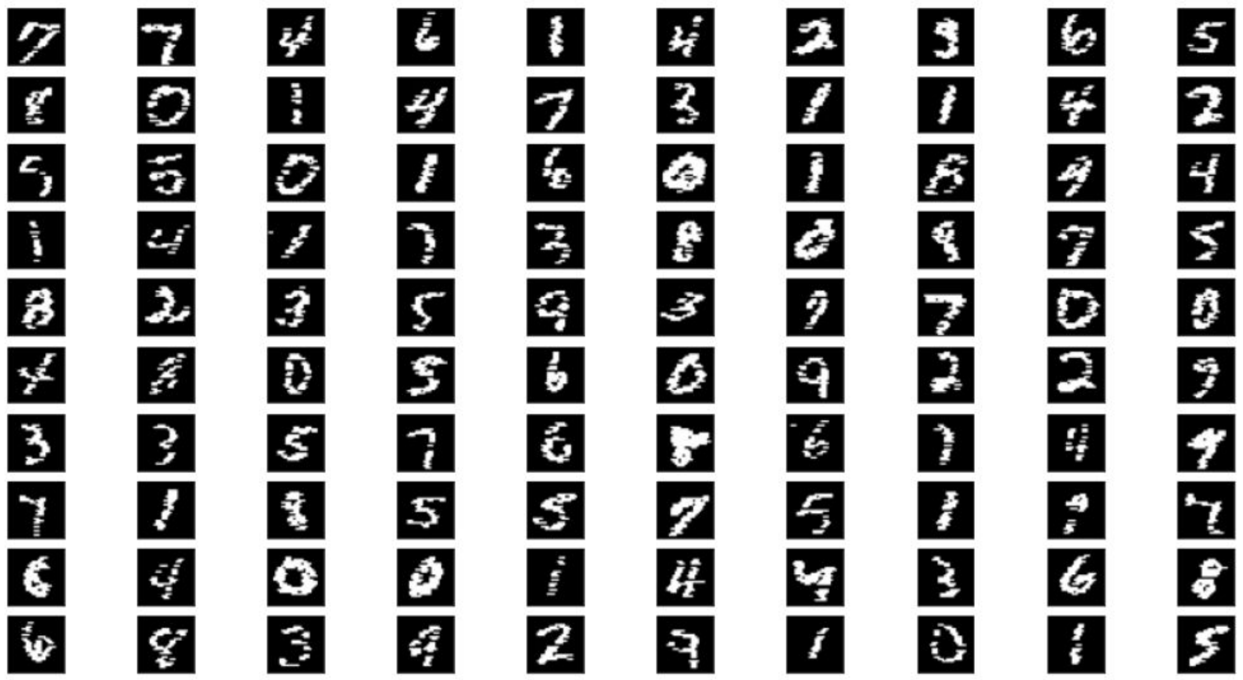
In order to reduce the noise and preserve image fidelity, we did the following steps:

1. Create a mask of the data by setting all number > MAX_PIXEL_VALUE * 65% to be 1, otherwise 0.
2. Apply Gaussian blur to the mask (with sigma = 0.7) in order to restore contiguous regions.
3. Apply masking of the blurred one more time such that all value > (MAX_MASK_VALUE - MEAN_MASK_VALUE)/8 + MEAN_MASK_VALUE are 1 and otherwise 0.

Here is the algorithm in python code:

```
def transform_data(data):
    mask = data > np.max(data)*0.65
    mask = mask.astype(float)
    for i in range(0, mask.shape[0]):
        mask[i] = gaussian_filter(mask[i], sigma=0.7)
    mask = mask > (np.max(mask) - np.mean(mask))/8 + np.mean(mask)
    mask = mask.astype(float)
    return mask
```

The result of the transformation is:



This is very close to the original MNIST dataset. However, there are still many other data processing choices, for example: standardization, normalization, principle component analysis, etc. In this project, we will explore standardization and principle component analysis effects to each machine learning models.

K-nearest neighbors (KNN):
In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor. Also *k*-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Support Vector Machine (SVM):
SVM is a supervised learning method where it is trying to draw a hyperplane or set of hyperplanes in a high- or infinite-dimensional space so that a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

Logistic Regression:
The logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This model is often use in a 2

class (binary) scenario. However, in a multiclass (N-class) classification situation, then N models are used to basically tell if an object belong to class X1 or not, X2 or not, X3 or not, etc.

Exploring data processing pipeline:
In order to get the best data processing steps, we perform training on 80% of the dataset (40,000 images) and testing on 20% (20,000 images). All models are used from sklearn package and at default parameters:

Model Name	Raw	Standardization Before	Denoise	Standardization After	PCA (40 components)	Preliminary accuracy
KNN	X					0.87158
KNN		X				0.81150
KNN			X			0.90292
KNN			X	X		0.86250
KNN			X		X	0.93925
SVM	X				X	0.95225
SVM			X		X	0.95917
LogReg	X					0.83508
LogReg		X				0.82817
LogReg			X			0.86900
LogReg			X	X		0.86650
LogReg			X		X	0.86058

From these results, it is concluded that the best data processing pipeline will include denoising and PCA with 40 components. The best number of components are found experimentally (ran with 30, 40, 50, 60, 100)

II. Hyperparameters grid-search

KNN:
For KNN, we select the 2 most impactful parameters: number of nearest neighbors (select from [1, 3, 4, 5,7, 9], and the distance metric [manhattan (1), euclid (2)]:
Test set score of (n=1, d=1): 0.92242
Test set score of (n=1, d=2): 0.92517
Test set score of (n=3, d=1): 0.92242

Test set score of (n=3, d=2): 0.92517
Test set score of (n=4, d=1): 0.92242
Test set score of (n=4, d=2): 0.92517
Test set score of (n=5, d=1): 0.92242
Test set score of (n=5, d=2): 0.92517
Test set score of (n=7, d=1): 0.92242
Test set score of (n=7, d=2): 0.92517
Test set score of (n=9, d=1): 0.92242
Test set score of (n=9, d=2): 0.92517

We concluded that d=2 (euclid) give the best accuracy on the test set no matter the number of nearest neighbors

SVM:
For SVM, we select the 2 most impactful parameters: kernel type (select from ['linear', 'poly', 'rbf', 'sigmoid'], and regularization parameter C [0.1, 0.5, 1, 2, 10]:
Test set score of (k=linear, c=0.1): 0.88317
Test set score of (k=linear, c=0.5): 0.88225
Test set score of (k=linear, c=1): 0.88233
Test set score of (k=linear, c=2): 0.88217
Test set score of (k=linear, c=10): 0.88225
Test set score of (k=poly, c=0.1): 0.93950
Test set score of (k=poly, c=0.5): 0.95392
Test set score of (k=poly, c=1): 0.95858
Test set score of (k=poly, c=2): 0.96025
Test set score of (k=poly, c=10): 0.95767
Test set score of (k=rbf, c=0.1): 0.93500
Test set score of (k=rbf, c=0.5): 0.95283
Test set score of (k=rbf, c=1): 0.95892
Test set score of (k=rbf, c=2): 0.96150
Test set score of (k=rbf, c=10): 0.96250
Test set score of (k=sigmoid, c=0.1): 0.81042
Test set score of (k=sigmoid, c=0.5): 0.75258
Test set score of (k=sigmoid, c=1): 0.73942
Test set score of (k=sigmoid, c=2): 0.73258
Test set score of (k=sigmoid, c=10): 0.72392

We concluded that k=rbf and C=10 give the best accuracy

LogReg:
For LogReg, we select the 2 most impactful parameters: penalty type (select from [L1, L2], and regularization parameter C [0.1, 0.5, 1, 2, 10]:
Test set score of (p=l1, c=0.1): 0.84867
Test set score of (p=l1, c=0.5): 0.84975

Test set score of (p=l1, c=1): 0.85000
Test set score of (p=l1, c=2): 0.84983
Test set score of (p=l1, c=10): 0.84967
Test set score of (p=l2, c=0.1): 0.84800
Test set score of (p=l2, c=0.5): 0.84942
Test set score of (p=l2, c=1): 0.84942
Test set score of (p=l2, c=2): 0.84967
Test set score of (p=l2, c=10): 0.84967

We concluded that p=L1=rbf and C=1 give the best accuracy.

III. Ensemble

In order to create an ensemble from the 3 machine learning models above, we use the VotingClassifier from sklearn.ensemble as follow:

```
cl1 = KNeighborsClassifier(n_neighbors=4, p=2)
cl2 = SVC(kernel='rbf', C=10.0, probability=True, max_iter=1000)
cl3 = LogisticRegression(penalty='l1', C=1.0, solver='liblinear')
eclf1 = VotingClassifier(estimators=[
    ('knn', cl1), ('svm', cl2), ('logreg', cl3)], voting='soft',
weights=[2, 3, 1], n_jobs=16)
eclf1.fit(X_train_trans_proj, y_train)
eclf1.score(X_test_trans_proj, y_test)
```

On the preliminary test, we achieved 0.96075, higher than LogReg and KNN but very slightly worse than SVM.

In order to evaluate the ensemble better, we cross-validate it using StratifiedKFold with number of splits = 4.

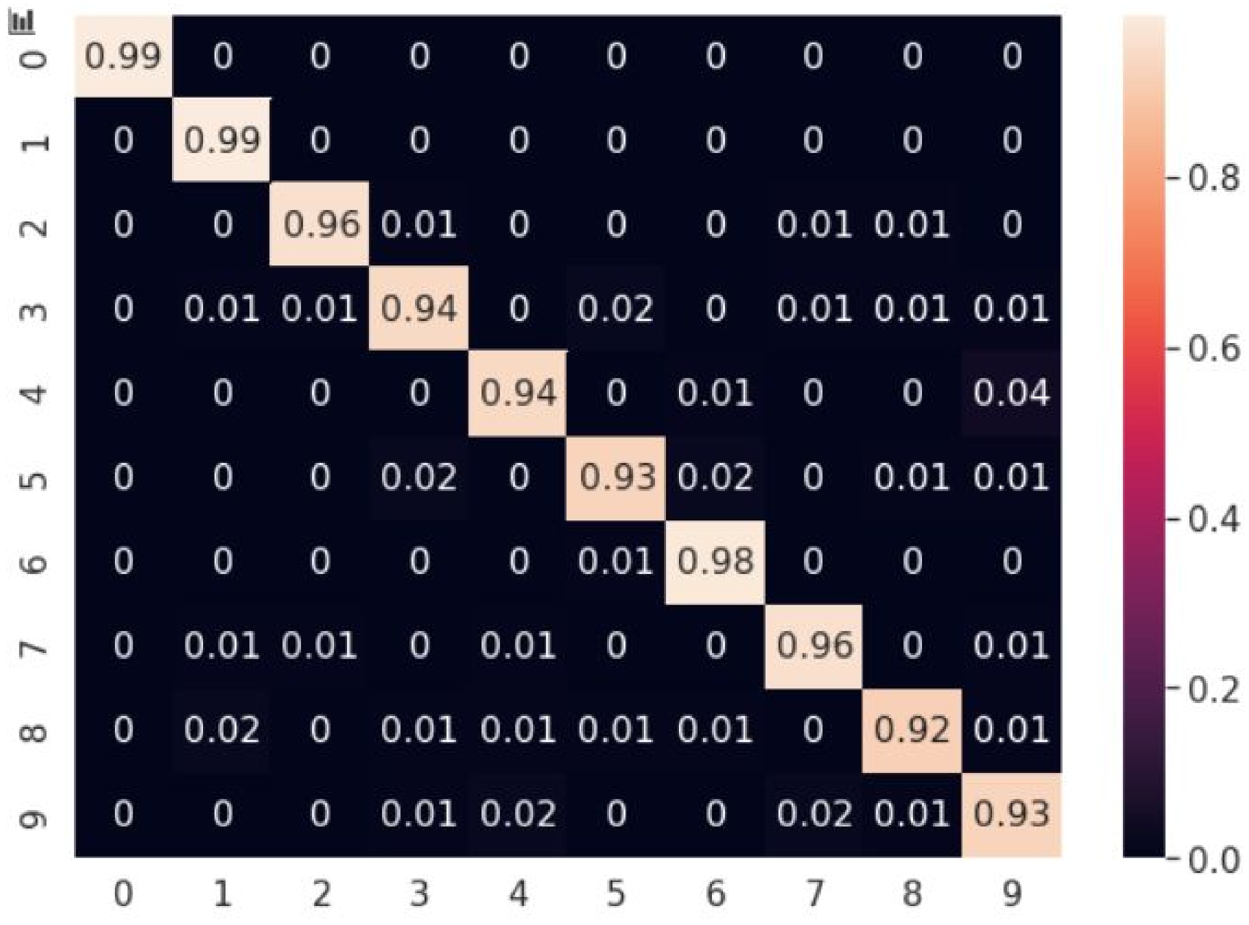
```
skf = StratifiedKFold(n_splits=4)
result = []
for train_idx, test_idx in skf.split(X, y):
    X_train_set = X[train_idx]
    X_test_set = X[test_idx]
    y_train_set = y[train_idx]
    y_test_set = y[test_idx]

    eclf1.fit(X_train_set, y_train_set)
    score = eclf1.score(X_test_set, y_test_set)
    print(score)
    result.append(score)
```

The results from 4 runs are:
[0.9604, 0.9578, 0.9601333333333333, 0.9554666666666667]
And the average accuracy is: 0.95845

This is the final accuracy score for our ensemble method.

Here is the confusion matrix for the ensemble:



0 0.99 0 0 0 0 0 0 0 0 0
1 0 0.99 0 0 0 0 0 0 0 0
2 0 0 0.96 0.01 0 0 0 0.01 0.01 0
3 0 0.01 0.01 0.94 0 0.02 0 0.01 0.01 0.01
4 0 0 0 0 0.94 0 0.01 0 0 0.04
5 0 0 0 0.02 0 0.93 0.02 0 0.01 0.01
6 0 0 0 0 0 0.01 0.98 0 0 0
7 0 0.01 0.01 0 0.01 0 0 0.96 0 0.01
8 0 0.02 0 0.01 0.01 0.01 0.01 0 0.92 0.01
9 0 0 0 0.01 0.02 0 0 0.02 0.01 0.93
0 1 2 3 4 5 6 7 8 9
0.8
0.6
0.4
0.2
0.0