Homework 2: Practicum

15 points total

Chang Yan (cyan13@jhu.edu), Jingguo Liang (jliang35@jhu.edu)

Instructions: This notebook is intended to guide you through creating and exploring your dataset. Please answer all questions in this notebook (you will see TODO annotations for where to include your answers). At the beginning of each part, we will bullet the expected deliverables for you to complete. All questions can be answered in 1-4 sentences, unless otherwise noted.

Part 1: Choosing a Dataset

Pick a dataset that you like (may be the same as hw1), but it should be within the **supervised learning** paradigm.

1) List the source of your dataset along with (very briefly) what you obtained from it.

For example:
Obtained features: 28*28 images from http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz.
Obtained labels from http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz.

My dataset is the same as homework 1, which is the MNIST dataset. I obtained 60000 28*28 images files of hand-written digits from them (They are in one binary file and I need to extract them). There is also another file containing the labels which is also a binary file I need to decode.

Part 2: Feature Engineering

If your data is not numerical, this will be difficult for an algorithm to learn directly. So, now that you've seen what the raw data looks like, you will start extracting *numerical* features from the raw data.

We obtain features through a process called **feature engineering**. Features may be derived from the existing raw data or may come from other data sources that can be associated with each example. This is a challenging task that often requires domain knowledge about the problem you are trying to solve.

For this question, **you will need to add a new feature to your dataset**. You will need some features for the other steps, but these can be very simple and don't need to rely on domain knowledge.

If your data is Wikipedia documents, possible features could be number of sentences, word count, the words that appear in the article, number of document revisions, number of contributing authors, number of references, etc. Notice that some of these features could be derived from the raw data (i.e. the words) while others may need to be downloaded separately (i.e. page metadata). If your

data are cat images, your features could be focus measure (i.e. blurriness/sharpness) using OpenCV Variance of Laplacian, whether image is grayscale, number of pixels, the pixel color values, etc. You can also use interaction terms, higher order terms, or indicators for ranges as your new feature.

You are free to obtain features in any way you like as long as you can justify why the features your propose should help solve the problem you've defined.

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [17]:
```
############################################################
# First read in the data as before
import cv2
import numpy as np
f = open('/content/drive/MyDrive/train-labels.idx1-ubyte', 'rb') # opening the label bina
content = f.read() # reading all lines
# the labels start at positon 8 and has total of 60000 ones
y = np.zeros(60000)
for i in range(60000):
  y[i] = content[i+8]
f = open('/content/drive/MyDrive/train-images.idx3-ubyte', 'rb') # opening the data binar
content = f.read() # reading all lines
# the images start at postion 16 and each is 28*28
X = np.zeros((60000, 28, 28))
for i in range(60000):
  temp = content[i*28*28+16: (i+1)*28*28+16]
  array = np.zeros(28*28)
  for j in range(28*28):
    array[j] = temp[j]
  X[i] = np.reshape(array,(28,28))

# compute new feature
X_flat = np.reshape(X, (60000, 28*28))
# feature 1: average intensity
feature1 = np.mean(X_flat, axis = 1)
feature1 = np.reshape(feature1, (60000, 1))
# feature 2: standard deviation of intensity
feature2 = np.std(X_flat, axis = 1)
feature2 = np.reshape(feature2, (60000, 1))
# feature 3: edge dectction of each picture
feature3 = np.zeros((60000, 28, 28))
ratio = 3 # use openCV recommendation
kernel_size = 3
low_threshold = 50
for i in range(X.shape[0]):
  X_blur = cv2.blur(np.uint8(X[i]), (3,3)) # follow openCV guidelines to blur first
  feature3[i] = cv2.Canny(X_blur, low_threshold, low_threshold*ratio, kernel_size)
feature3 = np.reshape(feature3, (60000, 28*28))

# Convert X and y to numpy arrays with appropriate dimensions
# stack all features
X = np.append(X_flat, feature1, axis = 1)
X = np.append(X, feature2, axis = 1)
X = np.append(X, feature3, axis = 1)
```

```
y = y
############################################################
# check the shape
print(X.shape)
print(y.shape)
```

```
(60000, 1570)
(60000,)
```

2) Describe the new features in your dataset.

I added 3 sets of features (a total of 1+1+28*28 = 786 features). The original features I used in last homework was only the image intensity, which is 28*28 = 784 features. This time I added 786 new features, so now it has 1570 features in total. The first set of feature is the mean intensity of each image. This is useful because this describes the overall intensity of each image, and images with more dark pixels will have higher average intensity. The second set of feature is the standard deviation of intensity of each image, this is useful beacuse it describes the level of dispersion in intensity of each image. The third set of features is the edge detection result of OpenCV Canny function. This is actually consists of 784 features because the edge detection result is a 28*28 1/0 binary array, so we need 784 features to describe the full distribution of edges. This set of features is useful beacuse edge detection is such a commonly used way in image preprocessing to extract the boundaries of objects in the picture.

Part 3: Evaluation: Usefulness of Added Feature

Now that you have added a new feature, train 2 logistic regression classifiers, one with the new feature and 1 without the new feature.

Choose at least 3 metrics that you have seen (ex. from Practicum 1) to evaluate and compare the performance of your 2 models.

In [ ]:

```
############################################################
# TODO: train 2 logistic regression classifiers. You may use libraries like sklearn for t
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# train_test split of two sets of features
# This is the set with new features
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25,
                                                    random_state=0)
# This is the set without new features
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_flat, y,
                                                            test_size=0.25,
                                                            random_state=0)
# model 1 with new features
clf = LogisticRegression(random_state=0, max_iter = 1000,
                         class_weight = 'balanced').fit(X_train, y_train)
y_test_hat = clf.predict(X_test)
# model 2 without new features
clf2 = LogisticRegression(random_state=0, max_iter = 1000,
                          class_weight = 'balanced').fit(X_train_2, y_train_2)
y_test_2_hat = clf2.predict(X_test_2)
############################################################
```

In [24]:
```
# evaluation of models
from sklearn.metrics import confusion_matrix
# with new features
# Use binary label to evaluate: label 1 is still 1, other lables are set to 0
# So the positive is detecting an "1" in image, the negative is all other numbers
y_test[y_test != 1] = 0
y_test_hat[y_test_hat != 1] = 0
y_test_2[y_test_2 != 1] = 0
y_test_2_hat[y_test_2_hat != 1] = 0
print("with new features:")
TN, FP, FN, TP = confusion_matrix(y_test, y_test_hat).ravel()
print("TP = "+str(TP))
print("FP = "+str(FP))
print("FN = "+str(FN))
print("TN = "+str(TN))
Accuracy = (TP + TN)/(TP + TN + FP + FN)
Precision = Sensitivity = TP/(TP + FP)
Specificity = TN/(TN + FP)
False_Positive_Rate = FP/(FP + TN)
print("Accuracy = "+str(Accuracy))
print("Precision = "+str(Precision))
print("Specificity = "+str(Specificity))
print("False Positive Rate = "+str(False_Positive_Rate))
# without new features
print("without new features:")
y_test_2_binary = y_test_2 # binarize
y_test_2_binary[y_test_2 != 1] = 0
y_test_pred_2_binary = y_test_2_hat # binarize
y_test_pred_2_binary[y_test_2_hat != 1] = 0
[TN2, FP2, FN2, TP2] = confusion_matrix(y_test_2, y_test_2_hat).ravel()
print("TP = "+str(TP2))
print("FP = "+str(FP2))
print("FN = "+str(FN2))
print("TN = "+str(TN2))
Accuracy2 = (TP2 + TN2)/(TP2 + TN2 + FP2 + FN2)
Precision2 = Sensitivity2 = TP2/(TP2 + FP2)
Specificity2 = TN2/(TN2 + FP2)
False_Positive_Rate2 = FP2/(FP2 + TN2)
print("Accuracy = "+str(Accuracy2))
print("Precision = "+str(Precision2))
print("Specificity = "+str(Specificity2))
print("False Positive Rate = "+str(False_Positive_Rate2))
```

```
with new features:
TP = 1674
FP = 86
FN = 54
TN = 13186
Accuracy = 0.9906666666666667
Precision = 0.9511363636363637
Specificity = 0.9935201928872816
False Positive Rate = 0.006479807112718505
without new features:
TP = 1693
FP = 184
FN = 35
TN = 13088
Accuracy = 0.9854
Precision = 0.9019712306872669
Specificity = 0.9861362266425557
False Positive Rate = 0.013863773357444244
```

3) Was your new feature helpful?

From the metrics above we can see that the new features are helpful. The accuracy increased from 0.9854 to 0.9907, the precision increased from 0.9020 to 0.9511, and the specificity increased from 0.9861 to 0.9935. The false positive rate decreased from 0.0139 to 0.0065, which is also an improvement. All of the four metrics above indicate that adding the new features are helpful. If we look at the TP, FP, FN and TN themselves, the false positive dramatically decreased, but the false negative increased a little. The true negative also increased largely, but the true positive decreased a little. Overall, there is still more improvement.

Part 4: Evaluation: With vs. Without Regularization

Come back to this section after you finish your programming assignment. Evaluate your logistic regression models (with and without regularization) from the programming assignment. Evaluate the models using the validation data we provided and compare the performance of the 2 models.

The first model without regularization get an accuracy of 0.83333(25/30), and converges at a log loss of about 0.465. The second model with regularization and $\lambda = 10$ get an accuracy of 0.90000(27/30), and converges at a log loss of about 0.484. They converge at roughly the same number of iterations. If we set a larger λ, the second model will converges slower but the accuracy does not increase. So we can conclude that the second model with regularization works better (has higher accuracy) than the first model, and it works best at $\lambda=10$ without affecting the number of iterations much.

4) Which model out of the two would you choose to deploy in real life?

I would choose to use the model with regularization, as it obviously improves accuracy and did not require much longer time to compute.

Submit

Great work! You're all done.

Make sure to submit this Python notebook (as a PDF) and the dataset you created as a zip file. See the homework writeup for directions.