Kosuke Takahashi
Liu
CS596
December 2nd, 2019

Homework Assignment 4 and 5 Write-up

Part I

I split the 60,000 training images into two more subsets using the following code:

```
dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset = dataset.repeat()
dataset = dataset.shuffle(buffer_size)
dataset = dataset.batch(batch_size)
iterator = dataset.make_one_shot_iterator()
```

Using the util.py function, func_confusion_matrix, the confusion matrix, average accuracy, precision and recall were calculated.

```
y_pred = model.predict(X_test)
conf_matrix, accuracy, recall_array, precision_array = are(y_test, y_pred)
```

Part II

Step 1 was done for us in the program so the data was already split.
For Step 2, the training set had to be evenly split into two more subsets which would be used for training and validation. The way I came up with this was similar to how the dataset was split in Step 1. But instead of splitting it into groups of 100, the dataset had to be split into two groups of 50.

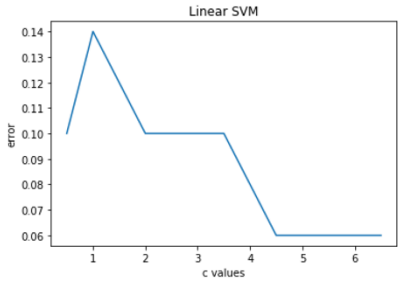
```
n2 = len(Xtr)
S2 = np.random.permutation(n2)
# subsets for training models
# has to be half of the 100 previous dataset
x_train= Xtr[S2[:50],:]
y_train= Ytr[S2[:50]]
# subsets for validation
x_validation= Xtr[S2[50:],:]
y_validation= Ytr[S2[50:]]
```

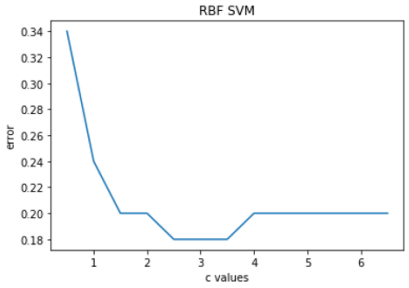
For Step 3, two figures had to be made. One figure used the selection of C's to train a SVM classifier. The classifiers were a linear SVM, a polynomial SVM, and a RBF SVM. The code to make the figure for linear, polynomial, and RBF was similar. The only difference was to change the 'linear' in:

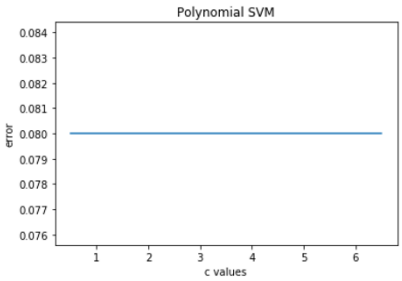
```
model = svm.SVC(kernel='linear', C=c_value)
```

The other figure was a selection of kernels.

The following are the figures for Figure 1 that were created using my values for C (Figure 1).



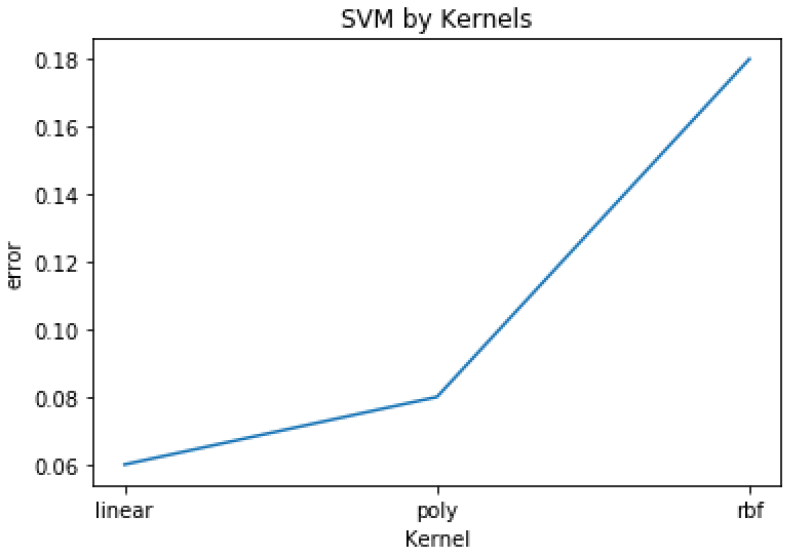





```
c_parameters = []
c_range =  np.arange(0.5,7.0,0.5)
svm_c_error = []
for c_value in c_range:
    model = svm.SVC(kernel='linear', C=c_value)
    model.fit(X=x_train, y=y_train)
    error = 1. - model.score(x_validation, y_validation)
    svm_c_error.append(error)
plt.plot(c_range, svm_c_error)
plt.title('Linear SVM')
plt.xlabel('c values')
plt.ylabel('error')
#plt.xticks(c_range)
plt.show()

# needs an index to keep track of the process
# c_parameters changes the data before going through the nest for loop
index = np.argmin(svm_c_error)
c_parameters.append(c_range[index])
```

The following is the figure for Figure 2 using kernel values




```
for kernel_value in kernel_types:
    # your own codes
    model = svm.SVC(kernel = kernel_value, C =c_parameters[x])
    model.fit(X=x_train, y=y_train)
    error =  1. - model.score(x_validation, y_validation)
    svm_kernel_error.append(error)
    x +=1
# similar to the for loop used for Figure 1 but x needs to be iterated
```

For Step 4, the best hyper-parameters were selected and applied over the testing subset.

```
best_kernel = kernel_types[best]
best_c = c_parameters[best] # poly had many that were the "best"
model = svm.SVC(kernel=best_kernel, C=best_c)
model.fit(X=x_train, y=y_train)
```

In Step 4, I had to find the best kernel and c value during the process of testing the subset. Then those values were stored into a model containing the testing dataset.

For Step 5, the results had to be evaluated using the best kernel, best c, confusion matrix, average accuracy, per-class precision and recall. Including this, there had to be 5 successful and 5 failed examples. Below is the results of running the process:

```
Best kernel: linear   c = 4.5
Confusion Matrix:
[[48  4]
 [ 1 47]]
Average Accuracy: 0.95
Per-Class Precision: [0.97959184 0.92156863]
Per-Class Recall: [0.92307692 0.97916667]

5 Successful Examples
Correct Prediction: 1.0
Features: [ 1.  15.1 13.8 31.7 36.6 13. ]
```

```
Correct Prediction: -1.0
Features: [ 0.  12.5  9.4 24.2 27.  11.2]
Correct Prediction: 1.0
Features: [ 0.  21.7 17.1 41.7 47.2 19.6]
Correct Prediction: -1.0
Features: [ 0.  15.7 12.2 31.7 34.2 14.2]
Correct Prediction: -1.0
Features: [ 0.  15.4 11.1 30.2 33.6 13.5]

5 Failure Examples
Prediction: -1.0    Ground-truth: 1.0
Features: [ 1.  19.2 16.5 40.9 47.9 18.1]
Prediction: -1.0    Ground-truth: 1.0
Features: [ 1.  16.4 14.  34.2 39.8 15.2]
Prediction: 1.0    Ground-truth: -1.0
Features: [ 1.   8.8  7.7 18.1 20.8  7.4]
Prediction: -1.0    Ground-truth: 1.0
Features: [ 0.  18.5 14.6 37.  42.  16.6]
Prediction: -1.0    Ground-truth: 1.0
Features: [ 0.  22.5 17.2 43.  48.7 19.8]
```