Allison Roeser

Kaggle username: aroeser

Digit Recognizer: Principal Components Analysis

Data Preparation, Exploration, Visualization:

The MNIST dataset contains 70,000 images of handwritten digits from 0 to 9. 42,000 of the images were provided a training that that included a label denoting which numerical digit each image represented. The test contained 28,000 unlabeled images. Figure 1 shows the first 5 observations of the train dataset. The first column is the label value and the other 784 columns represent the 784 pixels present on the the square image (28 pixels by 28 pixels). These 784 columns are binary to denote if the handwritten digit is contained in the pixel (value of 1) or if the pixel is bland (value of 0). Figure 2 & 3 show various images for the numbers 4 & 5. The digits are usually found centered in the image with blank space surrounding each digit. The amount of whitespace varies with each image.

Figure 4 displays the frequency of each digit in the training set. Digit 1 is most frequent with 4684 observations, and digit 5 is least frequent with 3,795 observations. The value counts for the digits are shown in Figure 5. Because each digit is roughly equally represented in the training set, frequency of occurrence should not have a strong impact on predicted value.

Implementation and Programming:

Full code is attached in the Appendix to show the specific implementation. Scikit-learn was leveraged for Random Forest Classification and Principal Component Analysis. Hyper-parameters such as criterion, number of estimators, and maximum depth were tested using Grid Search. Training, validation, and test sets were utilized to verify that the models generalized well. Confusion matrixes to were used to evaluate the classification accuracy of potential models. A Scree plot was used to illustrate the cumulative explained variance by the number of principle components. Pandas and Seaborn were leveraged for exploratory data analysis and visualization.

Review Research Design and Modeling Methods:

Initially, a Random Forest Classification model was fit to the training set using all 784 of the explanatory, predictor variables (representing the 784 pixels in the image). Grid search was used to determine the optimal hyperparameters, and the time to fit the model was noted. This Random Forest model was then used on the test set to predict the digit label for each of the 28,000 test set observations. These predictions were submitted to Kaggle, and Kaggle provided a ranking score.

Principal Component Analysis was then performed on the entire dataset (training and test) to reduce the number of variables. The time to identify the Principal Components was recorded. A second Random Forest Classification model was then run on the training data using the Principal Components identified, and the time to fit this model was noted. The new Random Forest Classification model was used with the training set, and the results were submitted to Kaggle.

Review Results, Evaluate Models:

The Random Forest model using all 784 explanatory variables had an accuracy score of 91.2% on the training set and 90.0% on the validation set (Figure 6 & 7). When submitted to Kaggle with the test set, the model achieved an accuracy score of 89.6% (Figure 8). Figure 9 displays the hyperparameters chosen using Grid Search.

Figure 10 & 11 contain the confusion matrices for the training and validation sets. The confusion matrices show that the model was most successful at classifying a digit 1 and least successful at classifying a digit 5. Actual 5s were frequently predicted to be 3s. The reverse effect was not seen. Most 3s were correctly identified as 3s (and not misclassified as 5s). Time to fit the Radom Forest Classification model on the full set of predictors using Grid Search was 50.2 seconds.

Principal Component Analysis was used to reduce the number of predictors to 154, which represents 95% of the explained variance present in the dataset. Figure 12 shows the Scree plot for the Cumulative Explained Variance by Number of Components. Fitting the PCA model took 18.9 seconds. The Random Forest classification model using the principal components was less accurate than the model using all 784 variables. The accuracy on training and validation was 84.0% and 80.9% respectively (Figure 13 & 14). The Kaggle score for the test set was 80.1% (Figure 15). Fitting the new Random Forest model took 14.4 seconds. When combined with the time to determine the principal components, the modeling process took 33.5 seconds, which is less than the 50.2 seconds the original model required. The confusion matrices for training and validation data are shown in Figures 16 & 17. 1 was still the digit most accurately predicted. Again, 5 was the digit most frequently predicted incorrectly. However, in the previous model 5 was usually mistaken for a 3. In this model, the digit predicted instead of 5 was more varied. The prediction was less precise.

Exposition, Problem Description and Management Recommendations:

Using the full set of 784 predictor variables produced a more accurate result than using the reduced number of predictors found using Principle Component Analysis. Although the time to run the Principle Component process and fit the Random Forest model on the reduced variables took less time than running the Random Forest model on the complete predictor set, the time difference was short. A longer computation time is a valid tradeoff for improved accuracy. If modeling took a substantial time period, then reducing the variables would be important. Similarly, if memory space for holding such a large number of predictors was problematic, reducing variable count would be important. In this application modeling is quick and not overly taxing on computing resources. It is recommended to use the full variable set for the most accurate prediction possible.

Appendix

Figure 1: First five rows of the training dataset

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...
0	1	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	...
2	1	0	0	0	0	0	0	0	0	0	...
3	4	0	0	0	0	0	0	0	0	0	...
4	0	0	0	0	0	0	0	0	0	0	...

Figure 2: Four images of the handwritten digit 4

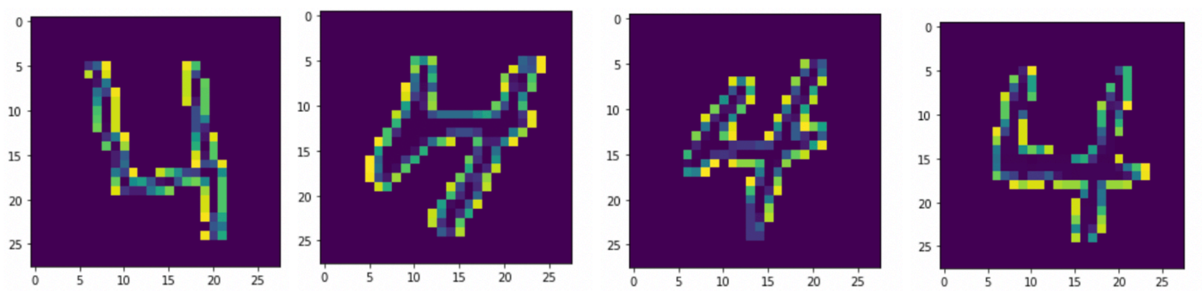


Figure 3: Four images of the handwritten digit 5

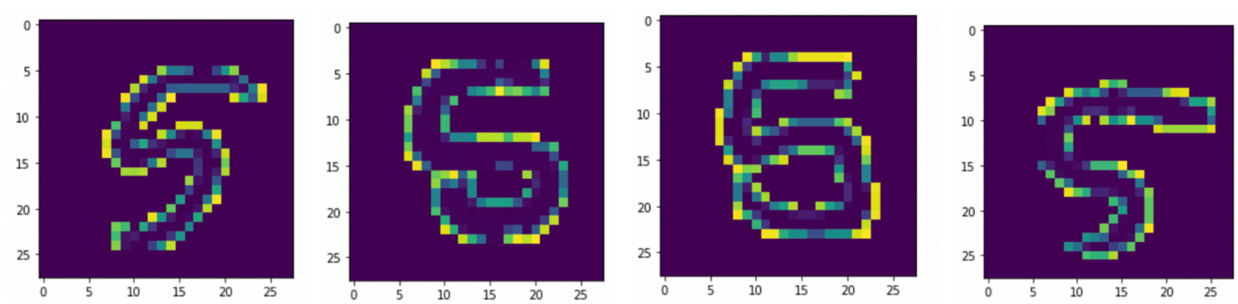


Figure 4: Frequency of each digit in the training set

Figure 5. Value counts of each digit in the training set

1	4684
7	4401
3	4351
9	4188
2	4177
6	4137
0	4132
4	4072
8	4063
5	3795

Figure 6. Random Forest model on Training Set: Accuracy, Precision, Recall and F1-score

Accuracy on training set: 0.912
f1: 0.9120697056806195

	precision	recall	f1-score	support
0	0.98	0.95	0.96	2526
1	0.98	0.91	0.94	3036
2	0.89	0.93	0.91	2397
3	0.88	0.89	0.88	2549
4	0.90	0.92	0.91	2387
5	0.86	0.94	0.90	2104
6	0.96	0.92	0.94	2578
7	0.92	0.92	0.92	2677
8	0.86	0.91	0.88	2304
9	0.89	0.85	0.87	2642
accuracy			0.91	25200
macro avg	0.91	0.91	0.91	25200
weighted avg	0.91	0.91	0.91	25200

Figure 7. Random Forest model on Validation Set: Accuracy, Precision, Recall and F1-score

Accuracy on validation set: 0.900
f1_vld: 0.8999663639586414

	precision	recall	f1-score	support
0	0.97	0.94	0.95	1708
1	0.98	0.91	0.94	1984
2	0.90	0.92	0.91	1646
3	0.85	0.88	0.86	1712
4	0.88	0.91	0.89	1587
5	0.84	0.91	0.87	1382
6	0.95	0.92	0.93	1724
7	0.91	0.91	0.91	1736
8	0.84	0.89	0.87	1524
9	0.89	0.83	0.86	1797
accuracy			0.90	16800
macro avg	0.90	0.90	0.90	16800
weighted avg	0.90	0.90	0.90	16800

Figure 8. Kaggle score for Random Forest model

Name	Submitted	Wait time	Execution time	Score
random forest.csv	just now	0 seconds	0 seconds	0.89585

Figure 9. Random Forest model best parameters from Grid Search

```
{'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 25}
```

Figure 10. Random Forest model on Training Set: Confusion Matrix

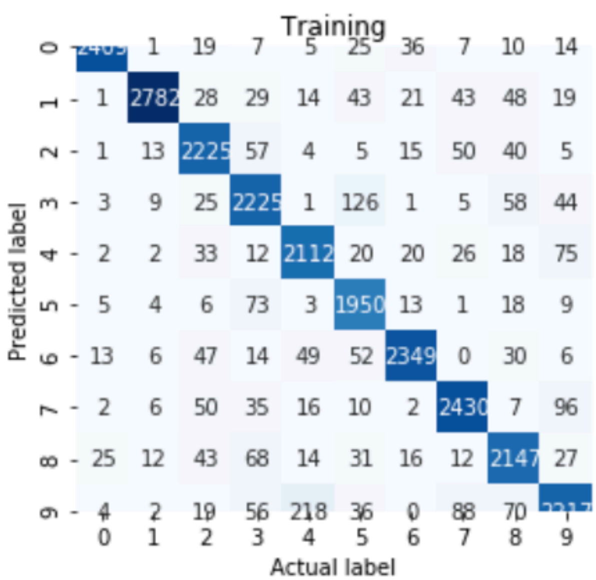


Figure 11. Random Forest model on Validation Set: Confusion Matrix

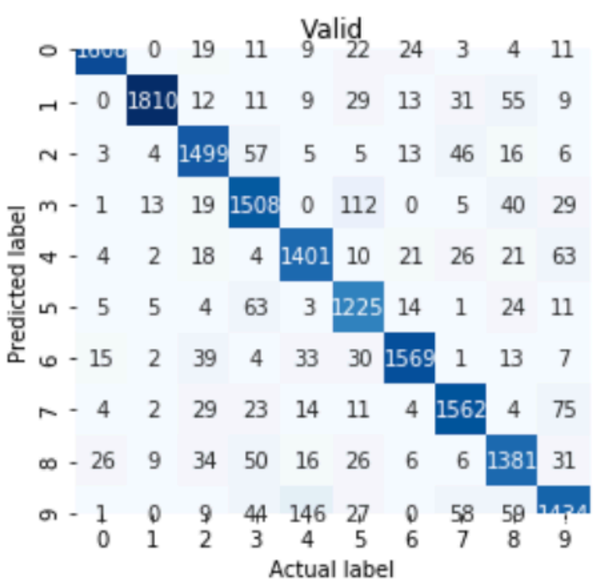


Figure 12. Scree plot displaying the cumulative explained variance by number of principal components

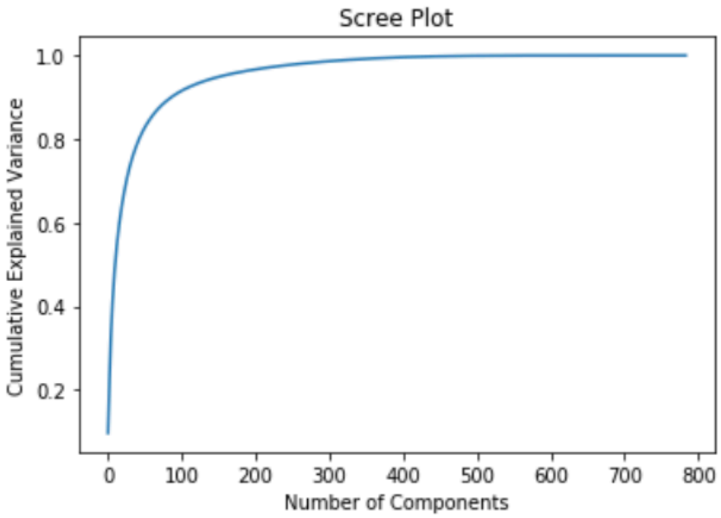


Figure 13. PCA / Random Forest model on Training Set: Accuracy, Precision, Recall and F1-score

```
Accuracy on training set: 0.840
f1: 0.8386199244889742
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      1964
           1       0.97      0.91      0.94      2377
           2       0.81      0.84      0.82      1885
           3       0.78      0.81      0.80      1942
           4       0.81      0.83      0.82      1847
           5       0.74      0.85      0.79      1557
           6       0.91      0.86      0.88      2036
           7       0.89      0.83      0.86      2254
           8       0.81      0.77      0.79      1999
           9       0.76      0.80      0.78      1879

    accuracy                           0.84     19740
   macro avg       0.84      0.84      0.84     19740
weighted avg       0.84      0.84      0.84     19740
```

Figure 14. PCA / Random Forest model on Validation Set: Accuracy, Precision, Recall and F1-score

```
Accuracy on training set: 0.840
f1: 0.8386199244889742
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      1964
           1       0.97      0.91      0.94      2377
           2       0.81      0.84      0.82      1885
           3       0.78      0.81      0.80      1942
           4       0.81      0.83      0.82      1847
           5       0.74      0.85      0.79      1557
           6       0.91      0.86      0.88      2036
           7       0.89      0.83      0.86      2254
           8       0.81      0.77      0.79      1999
           9       0.76      0.80      0.78      1879

    accuracy                           0.84     19740
   macro avg       0.84      0.84      0.84     19740
weighted avg       0.84      0.84      0.84     19740
```

Figure 15. Kaggle score for PCA and Random Forest model

Name	Submitted	Wait time	Execution time	Score
pca random forest.csv	just now	0 seconds	0 seconds	0.80114

Figure 16. PCA / Random Forest model on Training Set: Confusion Matrix

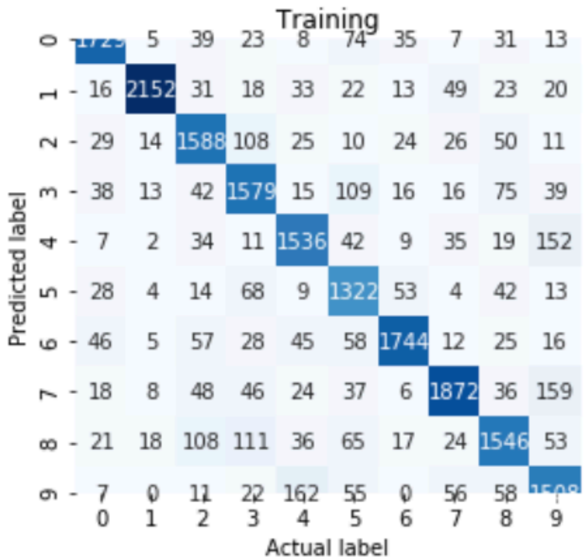
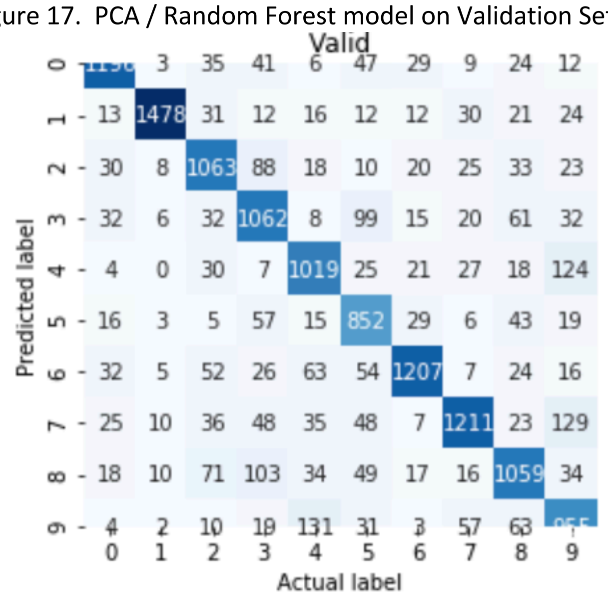


Figure 17. PCA / Random Forest model on Validation Set: Confusion Matrix



In [227]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from sklearn import metrics
import time
from sklearn.metrics import roc_auc_score, confusion_matrix, mean_squared_e
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
```

In [228]:
```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
#labels = pd.DataFrame(data = train, columns = ["label"])
#images = pd.DataFrame(train)
#images = images.drop(label, axis = 1)
```

In [ ]:
```
# Code from Chris' TA session utlized and modified
```

EDA

In [229]:
```
train_df.shape
```

Out[229]: (42000, 785)

In [230]:
```
test_df.shape
```

Out[230]: (28000, 784)

In [231]:
```
train_df.head()
```

Out[231]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775
0	1	0	0	0	0	0	0	0	0	0	...	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0

5 rows × 785 columns

In [232]:
```
#Split training data
X_train, X_valid, y_train, y_valid = train_test_split(train_df.drop(['label
                                                      train_size=.6, random_s
# Check the shape of the trainig data set array
print('Shape of X_train_data:', X_train.shape)
print('Shape of y_train_data:', y_train.shape)
print('Shape of X_valid_data:', X_valid.shape)
print('Shape of y_valid_data:', y_valid.shape)
```

```
Shape of X_train_data: (25200, 784)
Shape of y_train_data: (25200,)
Shape of X_valid_data: (16800, 784)
Shape of y_valid_data: (16800,)
```

In [233]:
```
y_test = pd.DataFrame(data = test_df, columns = ["label"])
```

In [234]:
```
X_test = pd.DataFrame(test_df)
```

In [235]:
```
X_train.head()
```

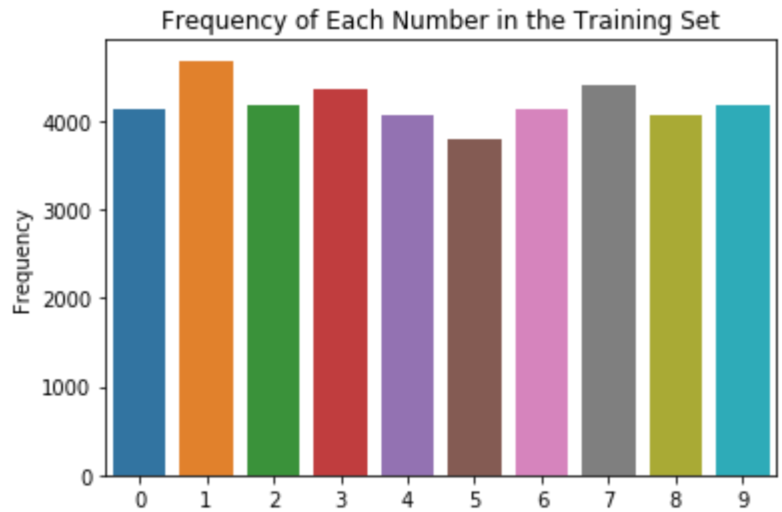
Out[235]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel'
25153	0	0	0	0	0	0	0	0	0	0	...	0	
14350	0	0	0	0	0	0	0	0	0	0	...	0	
24843	0	0	0	0	0	0	0	0	0	0	...	0	
6282	0	0	0	0	0	0	0	0	0	0	...	0	
41796	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 784 columns

In [236]:
```
lab_count = list(train_df.groupby('label')['label'].count())
index = list(train_df.groupby('label')['label'].count().index)

lab_plot = sns.barplot(y=lab_count,
                       x = index)
plt.ylabel("Frequency")
plt.title("Frequency of Each Number in the Training Set")
```

Out[236]:
```
Text(0.5, 1.0, 'Frequency of Each Number in the Training Set')
```



In [237]:
```
freq = train_df["label"].value_counts()
freq
```

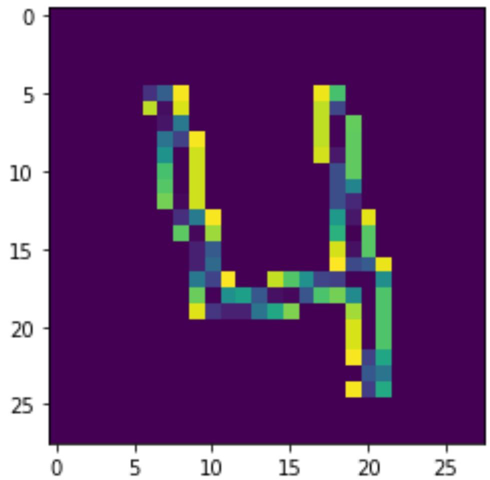
Out[237]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

In [238]:
```
train_4 = train_df[train_df['label'] == 4]
```

In [239]:
```
train_5 = train_df[train_df['label'] == 5]
```

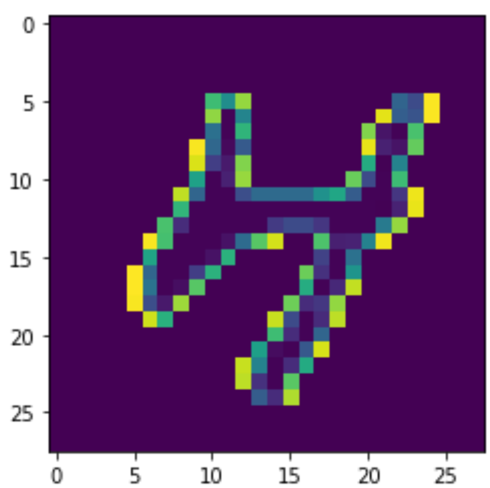
In [240]:
```
def gen_image(arr):
    two_d = (np.reshape(arr, (28,28)) * 255).astype(np.uint8)
    plt.imshow(two_d, interpolation = 'nearest')
    return plt

gen_image(train_4.iloc[0,1:].values)
```

Out[240]: <module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>



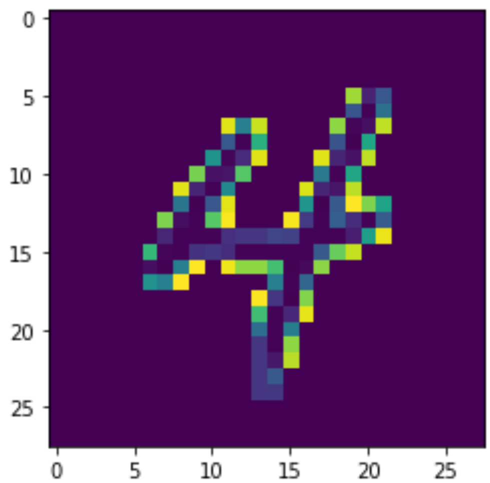
In [241]:
```
gen_image(train_4.iloc[1,1:].values)
```

Out[241]: <module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>



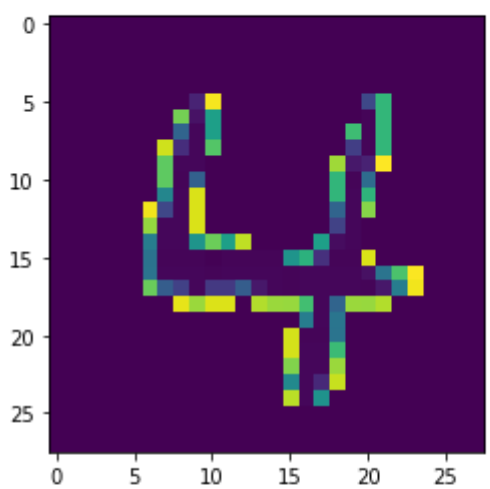
In [242]:
```
gen_image(train_4.iloc[2,1:].values)
```

Out[242]: <module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>



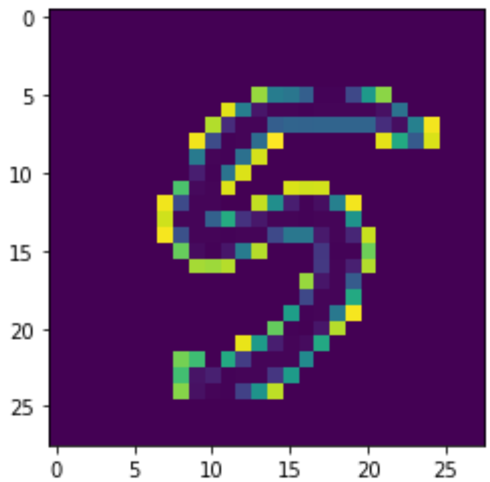
In [243]:
```
gen_image(train_4.iloc[3,1:].values)
```

Out[243]: <module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>



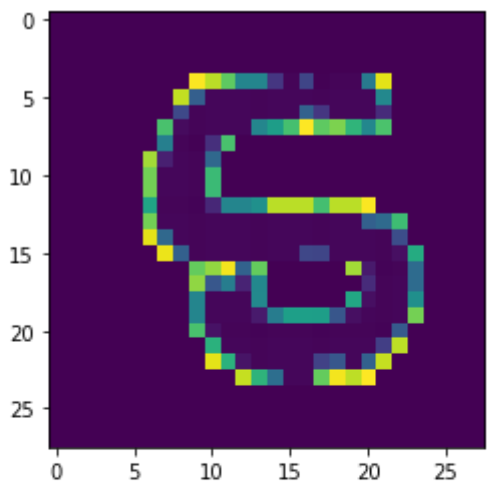
In [244]:
```
gen_image(train_5.iloc[0,1:].values)
```

Out[244]: <module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>



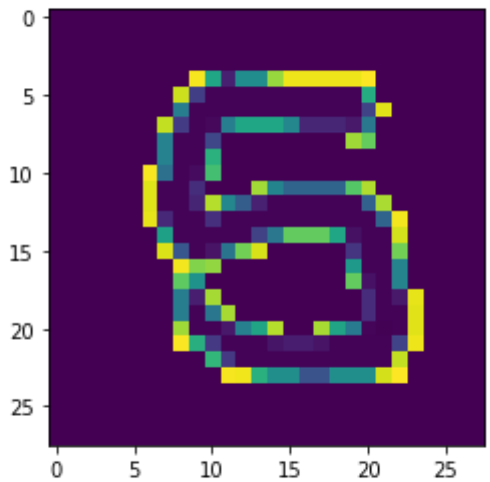
In [245]:
```
gen_image(train_5.iloc[1,1:].values)
```

Out[245]: <module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>



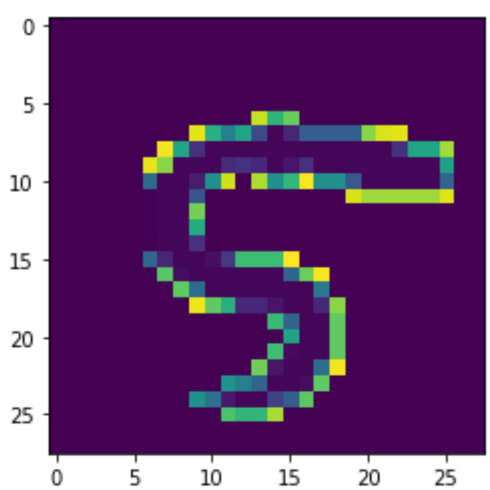
In [246]:
```
gen_image(train_5.iloc[2,1:].values)
```

Out[246]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



In [247]:
```
gen_image(train_5.iloc[3,1:].values)
```

Out[247]:
```
<module 'matplotlib.pyplot' from '/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py'>
```



Random Forest Model

In [289]:
```
start_time = time.process_time()
x = []

rfc = RandomForestClassifier()

param_grid = {
    'criterion': ['entropy', 'gini'],
    'max_depth': [3,5,7],
    'n_estimators': [10, 25]
}


rfc = GridSearchCV(estimator = rfc, param_grid = param_grid, n_jobs = 1)

#rfc = rfc.fit(X_train, y_train)

#rfc = RandomForestClassifier(n_estimators=30, n_jobs=-1, max_depth=5, crit
                             #max_features='sqrt', oob_score=True,  bootstr
# Train
rfc= rfc.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(rfc.score(X_train, y_train)
end_time = time.process_time()
f1 = f1_score(y_train, rfc.predict(X_train),average='weighted')
print("f1: {:}".format(f1))
# Extract single tree
print(metrics.classification_report(rfc.predict(X_train), y_train))


print("Accuracy on validation set: {:.3f}".format(rfc.score(X_valid, y_vali
f1_vld = f1_score(y_valid, rfc.predict(X_valid),average='weighted')
print("f1_vld: {:}".format(f1_vld))


print(metrics.classification_report(rfc.predict(X_valid), y_valid))

runtime = end_time - start_time  # seconds of wall-clock time
print(runtime)  # report in seconds
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/sklearn/mo
del_selection/_split.py:1978: FutureWarning: The default value of cv will
change from 3 to 5 in version 0.22. Specify it explicitly to silence this
warning.
  warnings.warn(CV_WARNING, FutureWarning)

Accuracy on training set: 0.912
f1: 0.9120697056806195
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      2526
           1       0.98      0.91      0.94      3036
           2       0.89      0.93      0.91      2397
           3       0.88      0.89      0.88      2549
           4       0.90      0.92      0.91      2387
           5       0.86      0.94      0.90      2104
           6       0.96      0.92      0.94      2578
           7       0.92      0.92      0.92      2677
```

```
           8       0.86      0.91      0.88      2304
           9       0.89      0.85      0.87      2642

    accuracy                           0.91     25200
   macro avg       0.91      0.91      0.91     25200
weighted avg       0.91      0.91      0.91     25200

Accuracy on validation set: 0.900
f1_vld: 0.8999663639586414
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      1708
           1       0.98      0.91      0.94      1984
           2       0.90      0.92      0.91      1646
           3       0.85      0.88      0.86      1712
           4       0.88      0.91      0.89      1587
           5       0.84      0.91      0.87      1382
           6       0.95      0.92      0.93      1724
           7       0.91      0.91      0.91      1736
           8       0.84      0.89      0.87      1524
           9       0.89      0.83      0.86      1797

    accuracy                           0.90     16800
   macro avg       0.90      0.90      0.90     16800
weighted avg       0.90      0.90      0.90     16800

50.19847000000004
```

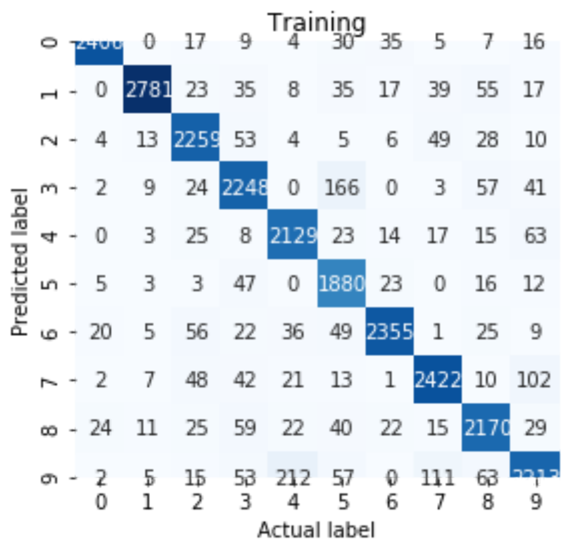
In [281]:
```
rfc.best_params_
```

Out[281]:
```
{'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 25}
```

In [249]:

```
#Confusion Matrix iRFC
cm_trn = confusion_matrix(y_train, rfc.predict(X_train))
cm_trn_plt=sns.heatmap(cm_trn.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Training");
fig1 = cm_trn_plt.get_figure()
fig1.show()
fig1.savefig('TrainCM.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.75, frameon=None)
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:8: UserWarning: Matplotlib is currently using module://ipyker
nel.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.

/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:12: MatplotlibDeprecationWarning:
The frameon kwarg was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use facecolor instead.
  if sys.path[0] == '':
```

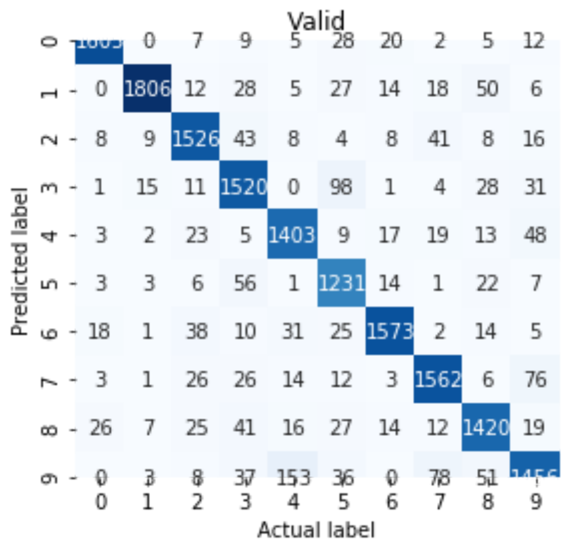


In [ ]:

In [ ]:

In [250]:
```
cm_vld = confusion_matrix(y_valid, rfc.predict(X_valid))
cm_vld_plt=sns.heatmap(cm_vld.T, square=True, annot=True, fmt='d', cbar=Fal
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Valid");
fig3 = cm_vld_plt.get_figure()
fig3.show()
fig3.savefig('ValidCM.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:7: UserWarning: Matplotlib is currently using module://ipyker
nel.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
  import sys
```



In [251]:
```
#Score test dataset
scr=rfc.predict(X_test)
#Conver array to Pandas dataframe with submission titles
pd_scr=pd.DataFrame(scr)
pd_scr.index.name = 'ImageId'
pd_scr.columns = ['Label']
print(pd_scr)
#Export to Excel
pd_scr.to_excel("rfr_only.xlsx")
```

```
         Label
ImageId
0            2
1            0
2            9
3            4
4            2
...        ...
27995        9
27996        7
27997        3
27998        9
27999        2

[28000 rows x 1 columns]
```

PCA

In [252]:
```
train_df_data = train_df.drop("label", axis = 1)
```

In [253]:
```
data = pd.concat([train_df_data, test_df])
data.head()
```

Out[253]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775
0	0	0	0	0	0	0	0	0	0	0	...	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0

5 rows × 784 columns

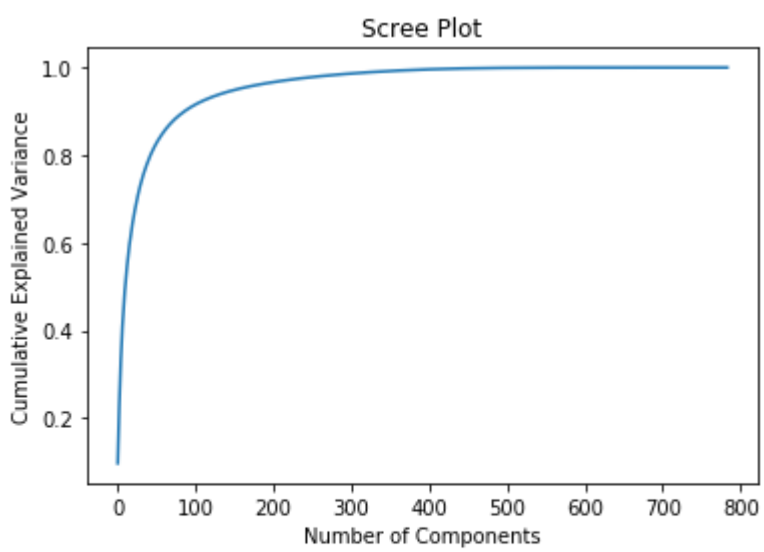
In [254]:
```
data.shape
```

Out[254]: (70000, 784)

In [255]:
```
#PCA on train and test set combined
```

In [256]:
```
#S8 Estimate number PCA components
start_time = time.clock()
pca = PCA().fit(data)
pca_plt=plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance');
plt.title("Scree Plot")
plt.savefig('PCAEstimate.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.5, frameon=None)
end_time = time.clock()
runtime = end_time - start_time
print(runtime)
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:2: DeprecationWarning: time.clock has been deprecated in Pyth
on 3.3 and will be removed from Python 3.8: use time.perf_counter or tim
e.process_time instead

/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:11: MatplotlibDeprecationWarning:
The frameon kwarg was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use facecolor instead.
  # This is added back by InteractiveShellApp.init_path()
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:12: DeprecationWarning: time.clock has been deprecated in Pyt
hon 3.3 and will be removed from Python 3.8: use time.perf_counter or tim
e.process_time instead
  if sys.path[0] == '':


18.896495000000016
```



In [257]:
```
CEVR = np.cumsum(pca.explained_variance_ratio_)
```

In [258]:
```
EVR = pca.explained_variance_ratio_
```

In [259]:
```
EVR_10 = EVR[0:10]
EVR_10
```

Out[259]:
```
array([0.09746116, 0.07155445, 0.06149531, 0.05403385, 0.04888934,
       0.04305227, 0.03278262, 0.02889642, 0.02758364, 0.0234214 ])
```

In [260]:
```
CEVR_10 = CEVR[0:10]
CEVR_10
```

Out[260]:
```
array([0.09746116, 0.16901561, 0.23051091, 0.28454476, 0.3334341 ,
       0.37648637, 0.40926898, 0.4381654 , 0.46574904, 0.48917044])
```

In [290]:
```
#S9 Keep the principal components that explain 95% of variation in the data
start_time = time.clock()
pca = PCA(n_components=0.95)
# fit PCA model to beast cancer data
pca.fit(data)
end_time = time.clock()
runtime_PCA = end_time - start_time  # seconds of wall-clock time

# transform data onto the first two principal components
X_pca = pca.transform(data)
print("Original shape: {}".format(str(data.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))

print(runtime_PCA)  # report in milliseconds
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:2: DeprecationWarning: time.clock has been deprecated in Pyth
on 3.3 and will be removed from Python 3.8: use time.perf_counter or tim
e.process_time instead

/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:6: DeprecationWarning: time.clock has been deprecated in Pyth
on 3.3 and will be removed from Python 3.8: use time.perf_counter or tim
e.process_time instead


Original shape: (70000, 784)
Reduced shape: (70000, 154)
19.108490999999958
```

In [ ]:

In [ ]:

In [264]:
```
totimages = X_pca
totimages.shape
```

Out[264]:
```
(70000, 154)
```

In [305]:
```
trainimages = totimages[0:42000, :]
testimages = totimages[42000:70000,:]
```

In [313]:

```
#S10 Split into train and validation prior to cross validation
X_pca_train, X_pca_vld, y_pca_train, y_pca_vld= train_test_split(trainimage
                                                                test_size=1380
print(X_pca_train.shape)
print(X_pca_vld.shape)
print(y_pca_train.shape)
print(y_pca_vld.shape)

#S11 Split Train and Test
X_pca_trn, X_pca_tst, y_pca_trn, y_pca_tst = train_test_split(X_pca_train,
                                                        test_size =0.3, random_
print(X_pca_trn.shape)
print(X_pca_tst.shape)
print(y_pca_trn.shape)
print(y_pca_tst.shape)

#S12 RF PCA Model
start_time = time.clock()

rfc = RandomForestClassifier()

param_grid = {
    'criterion': ['entropy'],
    'max_depth': [3,5,7],
    'n_estimators': [10]
}


rfc_pca = GridSearchCV(estimator = rfc, param_grid = param_grid, n_jobs = 1

#rfc_pca = RandomForestClassifier(n_estimators=10, n_jobs=-1, max_depth=5,
                                #max_features='sqrt', oob_score=True,  boot
# Train
rfc_pca= rfc_pca.fit(X_pca_trn, y_pca_trn)
print("Accuracy on training set: {:.3f}".format(rfc_pca.score(X_pca_trn, y_

f1 = f1_score(y_pca_trn, rfc_pca.predict(X_pca_trn),average='weighted')
f1_tst = f1_score(y_pca_tst, rfc_pca.predict(X_pca_tst),average='weighted')
f1_vld = f1_score(y_pca_vld, rfc_pca.predict(X_pca_vld),average='weighted')

print("f1: {:}".format(f1))


# Extract single tree
print(metrics.classification_report(rfc_pca.predict(X_pca_trn), y_pca_trn))

print("Accuracy on validation set: {:.3f}".format(rfc_pca.score(X_pca_tst,
print(metrics.classification_report(rfc_pca.predict(X_pca_vld), y_pca_vld))
print("f1_vld: {:}".format(f1_vld))

end_time = time.clock()
runtime_rf2 = end_time - start_time  # seconds of wall-clock time
print(runtime_rf2)  # report in milliseconds
```

```
(28200, 154)
(13800, 154)
(28200,)
```

```
(13800,)
(19740, 154)
(8460, 154)
(19740,)
(8460,)

/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:18: DeprecationWarning: time.clock has been deprecated in Pyt
hon 3.3 and will be removed from Python 3.8: use time.perf_counter or tim
e.process_time instead
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/sklearn/mo
del_selection/_split.py:1978: FutureWarning: The default value of cv will
change from 3 to 5 in version 0.22. Specify it explicitly to silence this
warning.
  warnings.warn(CV_WARNING, FutureWarning)

Accuracy on training set: 0.840
f1: 0.8386199244889742
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      1964
           1       0.97      0.91      0.94      2377
           2       0.81      0.84      0.82      1885
           3       0.78      0.81      0.80      1942
           4       0.81      0.83      0.82      1847
           5       0.74      0.85      0.79      1557
           6       0.91      0.86      0.88      2036
           7       0.89      0.83      0.86      2254
           8       0.81      0.77      0.79      1999
           9       0.76      0.80      0.78      1879

    accuracy                           0.84     19740
   macro avg       0.84      0.84      0.84     19740
weighted avg       0.84      0.84      0.84     19740

Accuracy on validation set: 0.809
              precision    recall  f1-score   support

           0       0.87      0.85      0.86      1402
           1       0.97      0.90      0.93      1649
           2       0.78      0.81      0.79      1318
           3       0.73      0.78      0.75      1367
           4       0.76      0.80      0.78      1275
           5       0.69      0.82      0.75      1045
           6       0.89      0.81      0.85      1486
           7       0.86      0.77      0.81      1572
           8       0.77      0.75      0.76      1411
           9       0.70      0.75      0.72      1275

    accuracy                           0.80     13800
   macro avg       0.80      0.80      0.80     13800
weighted avg       0.81      0.80      0.81     13800

f1_vld: 0.8027605434505156
14.439937999999984

/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykerne
l_launcher.py:51: DeprecationWarning: time.clock has been deprecated in
```

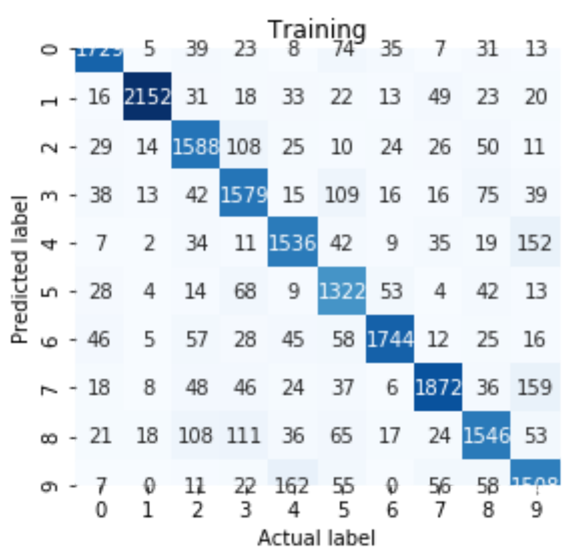
```
Python 3.3 and will be removed from Python 3.8: use time.perf_counter o
r time.process_time instead
```

In [314]:
```python
runtime_total = runtime_PCA + runtime_rf2
runtime_total
```

Out[314]:
```
33.54842899999994
```

In [317]:
```python
#Confusion Matrix RFC-PCA
cm_trn_pca = confusion_matrix(y_pca_trn, rfc_pca.predict(X_pca_trn))
cm_trn_pca_plt=sns.heatmap(cm_trn_pca.T, square=True, annot=True, fmt='d',
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Training");
fig4 = cm_trn_pca_plt.get_figure()
fig4.show()
fig4.savefig('TrainPCACM.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```

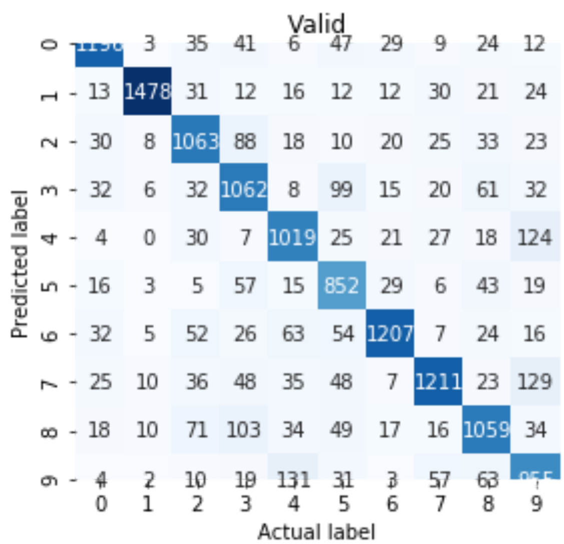
```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:8: UserWarning: Matplotlib is currently using module://ipyker
nel.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
```



In [319]:

```
cm_vld_pca = confusion_matrix(y_pca_vld, rfc_pca.predict(X_pca_vld))
cm_vld_pca_plt=sns.heatmap(cm_vld_pca.T, square=True, annot=True, fmt='d',
plt.xlabel('Actual label')
plt.ylabel('Predicted label')
plt.title("Valid");
fig6 = cm_vld_pca_plt.get_figure()
fig6.show()
fig6.savefig('ValidPCACM.png',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25)
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:7: UserWarning: Matplotlib is currently using module://ipyker
nel.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
  import sys
```



In [316]:
```
#Score test dataset
scr=rfc_pca.predict(testimages)
#Conver array to Pandas dataframe with submission titles
pd_scr=pd.DataFrame(scr)
pd_scr.index.name = 'ImageId'
pd_scr.columns = ['label']
print(pd_scr)
#Export to Excel
pd_scr.to_excel("rfc_pca.xlsx")
```

```
         label
ImageId
0            2
1            0
2            8
3            2
4            2
...        ...
27995        9
27996        7
27997        3
27998        9
27999        2

[28000 rows x 1 columns]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: