```
## set up notebook to display multiple output in one cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
import tensorflow as tf
from tensorflow import keras

from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import metrics

from keras.models import Sequential
from keras.layers import Dense , Dropout , Lambda, Flatten
from keras import  backend as K
from keras.preprocessing.image import ImageDataGenerator

from datetime import datetime
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
sns.set()
pd.set_option('display.max_rows', None)

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train.columns
train.info()
train.shape

Index(['label', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4',
'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778',
'pixel779',
```

```
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=785)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

```
(42000, 785)
```

```
test.columns
test.info()
```

```
Index(['pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
'pixel6',
       'pixel7', 'pixel8', 'pixel9',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778',
'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=784)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 167.5 MB
```

```
train.isnull().any().describe()
test.isnull().any().describe()
```

```
count       785
unique        1
top       False
freq        785
dtype: object
```

```
count       784
unique        1
top       False
freq        784
dtype: object
```

```
train_df_X = train.copy()
train_df_y = train_df_X['label']
train_df_X.drop(['label'], axis=1, inplace=True)
```

```
train_df_X.shape
```

```
train_df_y.shape
```

```
(42000, 784)
```

```
(42000,)
```

```
train_df_X_img = train_df_X.values.reshape(-1,28,28,1)

num_examples = 20
plt.figure(figsize=(20,20))
for i in range(num_examples):
    plt.subplot(1, num_examples, i+1)
    plt.imshow(train_df_X_img[i], cmap='Greys')
    plt.axis('off')
plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edecd7e610>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edec8d4250>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edec8afdc0>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edec8aaa30>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1ed932a4130>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1ed932a43d0>
```

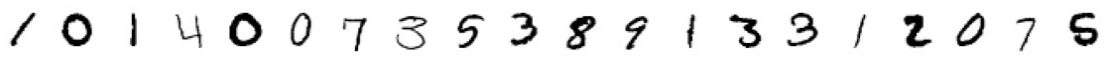
```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

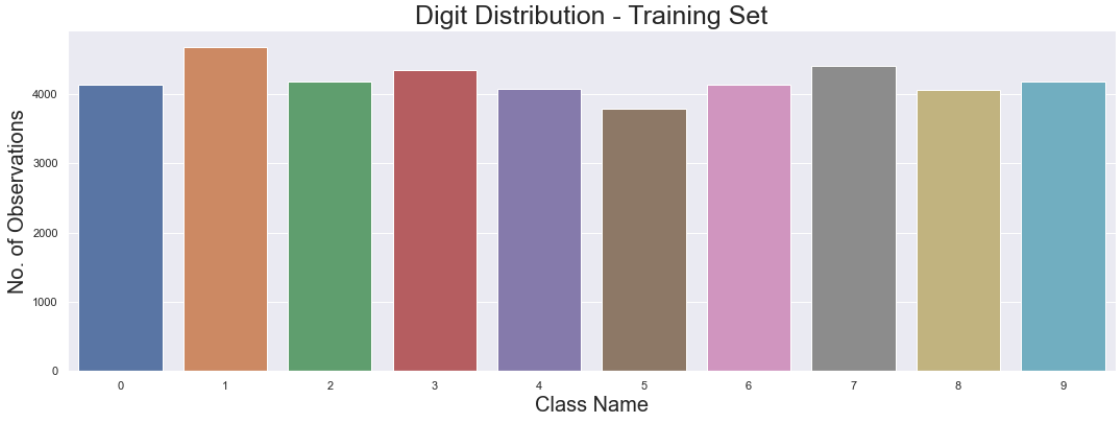
```
<matplotlib.image.AxesImage at 0x1ed93300f70>
```

```
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1ed9333d820>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1ed9336fee0>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1ed933a9760>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1ed933d7e20>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1ed93412760>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edecd6c4f0>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edec8e25e0>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edec8aa880>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1ed9333d250>
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1ed934fd190>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1ed93526940>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1ed93526fa0>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1ed935927c0>

(-0.5, 27.5, 27.5, -0.5)
```



```
ax = plt.subplots(figsize=(18, 6))
sns.set_style("whitegrid")
sns.countplot(x='label', data=train);
plt.ylabel("No. of Observations", size=20);
plt.xlabel("Class Name", size=20);
plt.title("Digit Distribution - Training Set", size=25)

<AxesSubplot:xlabel='label', ylabel='count'>

Text(0, 0.5, 'No. of Observations')

Text(0.5, 0, 'Class Name')

Text(0.5, 1.0, 'Digit Distribution - Training Set')
```



```
X_test_csv = test.values
```

```
X_test_csv
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Modelling

```
X_train,X_val,y_train,y_val=train_test_split(train_df_X,train_df_y,tes
t_size=0.2,random_state=42)
```

Random Forest Classifier

```
X_train = train.iloc[:,1:]
y_train = train.iloc[:,0]
```

```
rfc = RandomForestClassifier(random_state=42, n_jobs=-1)
start = datetime.now()
rfc.fit(X_train, y_train)
end = datetime.now()
print('Model Fit Timer:', end-start)
```

```
RandomForestClassifier(n_jobs=-1, random_state=42)
```

```
Model Fit Timer: 0:00:07.621886
```

```
start=datetime.now()
y_pred = rfc.predict(X_val)
end=datetime.now()
print(end-start)
print('accuracy score = ', accuracy_score(y_val,y_pred))
```

```
0:00:00.182363
accuracy score =  1.0
```

```
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       816
           1       1.00      1.00      1.00       909
           2       1.00      1.00      1.00       846
           3       1.00      1.00      1.00       937
           4       1.00      1.00      1.00       839
           5       1.00      1.00      1.00       702
           6       1.00      1.00      1.00       785
```

```
           7       1.00      1.00      1.00       893
           8       1.00      1.00      1.00       835
           9       1.00      1.00      1.00       838

    accuracy                           1.00      8400
   macro avg       1.00      1.00      1.00      8400
weighted avg       1.00      1.00      1.00      8400
```

```
X_val_img = X_val.values.reshape(-1,28,28,1)

num_examples = 20
plt.figure(figsize=(20,20))
for i in range(num_examples):
    plt.subplot(1, num_examples, i+1)
    plt.imshow(X_val_img[i], cmap='Greys')
    plt.axis('off')
plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1edbd6c5e50>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1edbd701550>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1edbd731940>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1edbd762fd0>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1edbd9cc7c0>

(-0.5, 27.5, 27.5, -0.5)

<AxesSubplot:>

<matplotlib.image.AxesImage at 0x1edbd9ccbb0>
```

```
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbda38700>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbda68e80>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdaa3640>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdad4dc0>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdb0f640>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdb3fd00>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdb7c490>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdbabca0>
(-0.5, 27.5, 27.5, -0.5)
<AxesSubplot:>
<matplotlib.image.AxesImage at 0x1edbdbbc100>
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edbdc18c10>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edbdc553a0>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edbdc84b20>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edbdcb6df0>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
<AxesSubplot:>
```

```
<matplotlib.image.AxesImage at 0x1edbdcf15e0>
```

```
(-0.5, 27.5, 27.5, -0.5)
```

```
y_pred[0:20]
```

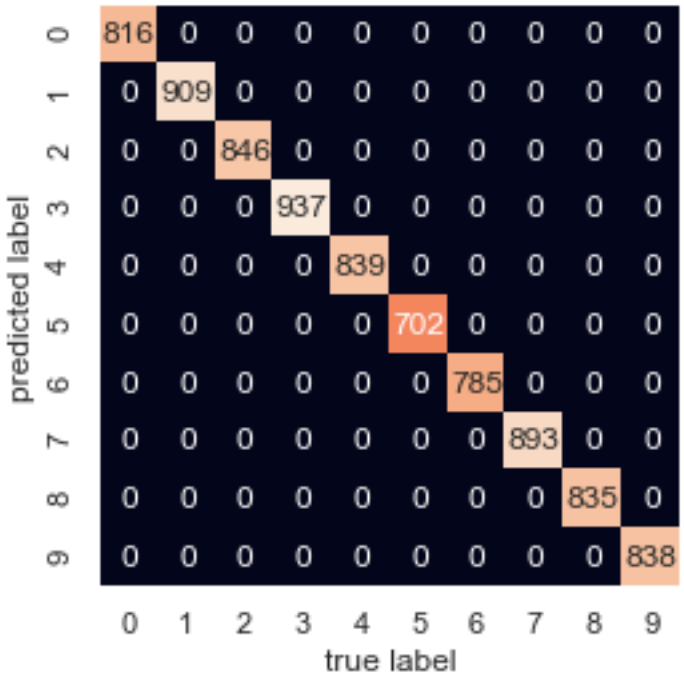
```
array([8, 1, 9, 9, 8, 6, 2, 2, 7, 1, 6, 3, 1, 2, 7, 4, 3, 3, 6, 4],
      dtype=int64)
```

```
y_val[0:20]
```

```
5457     8
38509    1
25536    9
31803    9
39863    8
30639    6
12986    2
41067    2
30743    7
6839     1
17164    6
21723    3
12272    1
5030     2
25222    7
```

```
34680    4
4976     3
19565    3
27947    6
31133    4
Name: label, dtype: int64
```

```
mat = confusion_matrix(y_val, y_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label');
```




```
start=datetime.now()
y = rfc.predict(test)
end=datetime.now()
print(end-start)
```

```
0:00:00.554716
```

```
# create submission file
submission = pd.DataFrame({"ImageId": (test.index + 1),"Label": y})
submission.to_csv('submission_rfc.csv', index=False)
```

```
from IPython.display import Image
Image(filename='submission_rfc.png')
```

YOUR RECENT SUBMISSION

submission_rfc.csv Score: 0.96575

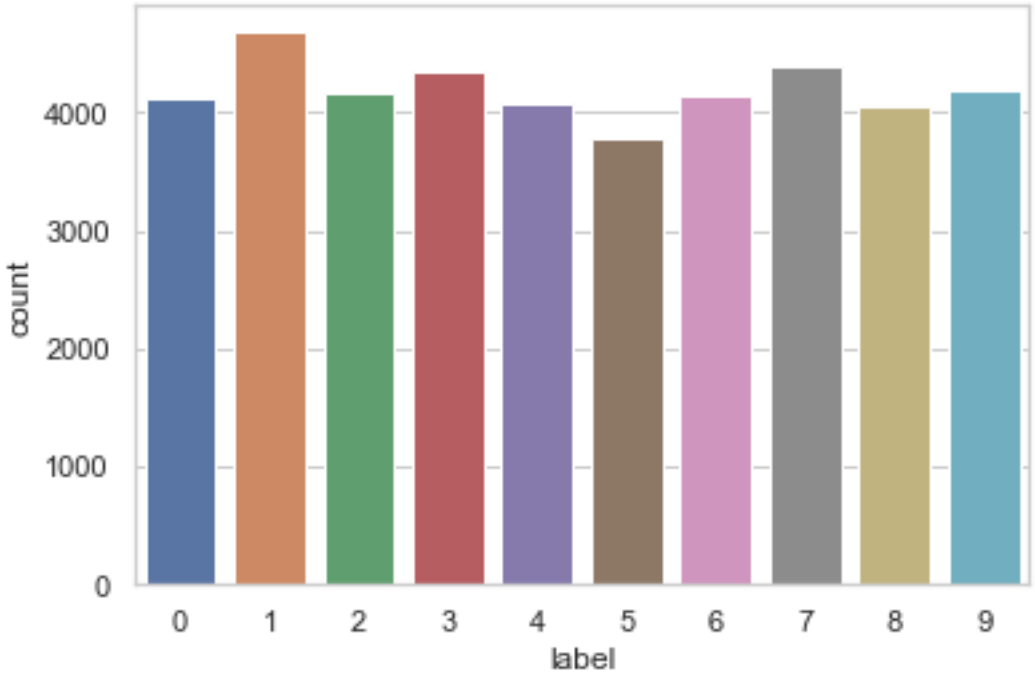
Submitted by SeafoodTakeout · Submitted just now

PCA

```
pca_train = train.drop('label', axis=1)

label_train = train.label

pca_alldata = pd.concat([pca_train, test], ignore_index=True)
pca_alldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 418.7 MB
```

```
L = sns.countplot(label_train)
```




```
start=datetime.now()
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
pca_n_components = np.argmax(cumsum >= 0.95) + 1
```

```
end=datetime.now()
print('Timer: ', end-start)
print('# PCA Components at 95% variability: ', pca_n_components)
```

```
PCA()
```

```
Timer:  0:00:09.184132
# PCA Components at 95% variability:  154
```

```
plt.figure(figsize=(6,4))
plt.plot(cumsum, linewidth=3)
plt.axis([0, 400, 0, 1])
plt.xlabel("Dimensions")
plt.ylabel("Explained Variance")
plt.plot([pca_n_components, pca_n_components], [0, 0.95], "k:")
plt.plot([0, pca_n_components], [0.95, 0.95], "k:")
plt.plot(pca_n_components, 0.95, "ko")
plt.annotate("Elbow", xy=(65, 0.85), xytext=(70, 0.7),
             arrowprops=dict(arrowstyle="->"), fontsize=16)
plt.grid(True)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

```
[<matplotlib.lines.Line2D at 0x1edbdfa9790>]
```

```
(0.0, 400.0, 0.0, 1.0)
```

```
Text(0.5, 0, 'Dimensions')
```

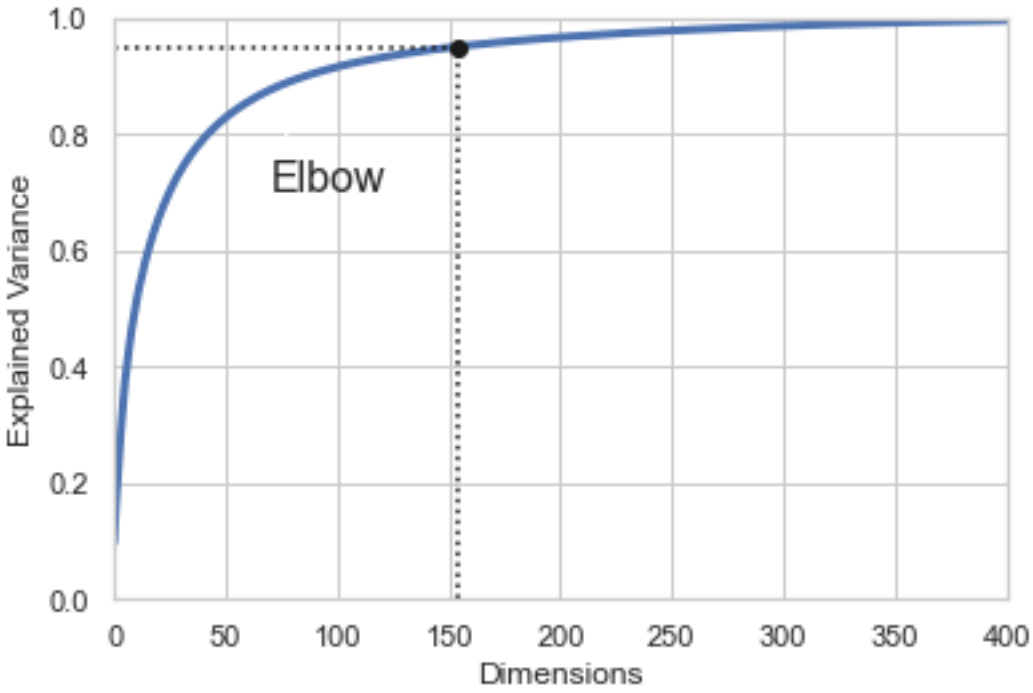
```
Text(0, 0.5, 'Explained Variance')
```

```
[<matplotlib.lines.Line2D at 0x1edbdfa9f70>]
```

```
[<matplotlib.lines.Line2D at 0x1edbdfa9fd0>]
```

```
[<matplotlib.lines.Line2D at 0x1edbdfb6370>]
```

```
Text(70, 0.7, 'Elbow')
```




```
start=datetime.now()
pca = PCA(n_components=pca_n_components)
X_train_pca = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_train_pca)
end=datetime.now()
print('Timer: ', end-start)
```

```
Timer:  0:00:04.491838
```

```
rnd_clf_pca = RandomForestClassifier(n_estimators=100,
random_state=42)

start=datetime.now()
rnd_clf_pca.fit(X_train_pca, y_train)
end=datetime.now()
print(end-start)
```

```
RandomForestClassifier(random_state=42)

0:02:36.257194
```

```
X_test_reduced = pca.transform(X_val)

y_pred_reduced = rnd_clf_pca.predict(X_test_reduced)
accuracy_score(y_val, y_pred_reduced)
```

```
1.0
```

```
X_test_reduced_csv = pca.transform(X_test_csv)
y_pred_reduced_csv = rnd_clf_pca.predict(X_test_reduced_csv)
```

```
# create submission file
submission2 = pd.DataFrame({"ImageId": (test.index + 1),"Label":
y_pred_reduced_csv})
submission2.to_csv('submission_pcs_rfc.csv', index=False)
```

```
from IPython.display import Image
Image(filename='submission_pcs_rfc.png')
```

K-Means Clustering

```
n_digits = len(np.unique(y_val))
print(n_digits)
```

```
10
```

```
start=datetime.now()
```

```
# Initialize Kmeans model
```

```
kmeans = MiniBatchKMeans(n_clusters = n_digits)
```

```
# Fit the model to the training data
```

```
kmeans.fit(X_train)
```

```
kmeans.labels_
```

```
end=datetime.now()
print(end-start)
```

```
MiniBatchKMeans(n_clusters=10)
```

```
array([1, 2, 4, ..., 8, 9, 5])
```

```
0:00:03.743961
```

```
y_train_clusters = pd.DataFrame(kmeans.labels_, columns=["Cluster"])
```

```
y_train_clusters['Label'] = y_train.tolist()
```

```
pd.crosstab(y_train_clusters['Cluster'],y_train_clusters["Label"])
```

```
Label       0     1     2     3     4     5     6     7     8     9
Cluster
0          78     4   194   508     4   807    55    28  2207    38
```

```
1           3  2240   424    71    72   223    79   226   212    65
2        2373     0    26    14     3    72    62    10    30    27
3        1380     0   320   182    15   349   224    14    39    16
4          11  2422   296   407   143   357   355   256   269   212
5          14     2    65    94  1957   205    35   565   139  1858
6          11     2    46    19  1782   426     7   387   131  1719
7         146     6   443  2980     0  1294    38     4   963    59
8           2     2    76    27     4     7     1  2902    24   178
9         114     6  2287    49    92    55  3281     9    49    16
```

```
key = y_train_clusters.groupby('Cluster').agg(lambda
x:x.value_counts().index[0])
key
```

```
         Label
Cluster
0            8
1            1
2            0
3            0
4            1
5            4
6            4
7            3
8            7
9            6
```

```
y_train_labels = []
for i in range(0,len(kmeans.labels_)):
    x = kmeans.labels_[i]
    y_train_labels += [key['Label'].loc[x]]

print(classification_report(y_train, y_train_labels))
```

```
              precision    recall  f1-score   support

           0       0.73      0.91      0.81      4132
           1       0.56      1.00      0.72      4684
           2       0.00      0.00      0.00      4177
           3       0.50      0.68      0.58      4351
           4       0.40      0.92      0.55      4072
           5       0.00      0.00      0.00      3795
           6       0.55      0.79      0.65      4137
           7       0.90      0.66      0.76      4401
           8       0.56      0.54      0.55      4063
           9       0.00      0.00      0.00      4188

    accuracy                           0.56     42000
   macro avg       0.42      0.55      0.46     42000
weighted avg       0.43      0.56      0.47     42000
```

```
y_val_pred = kmeans.predict(X_val)
y_val_pred
```

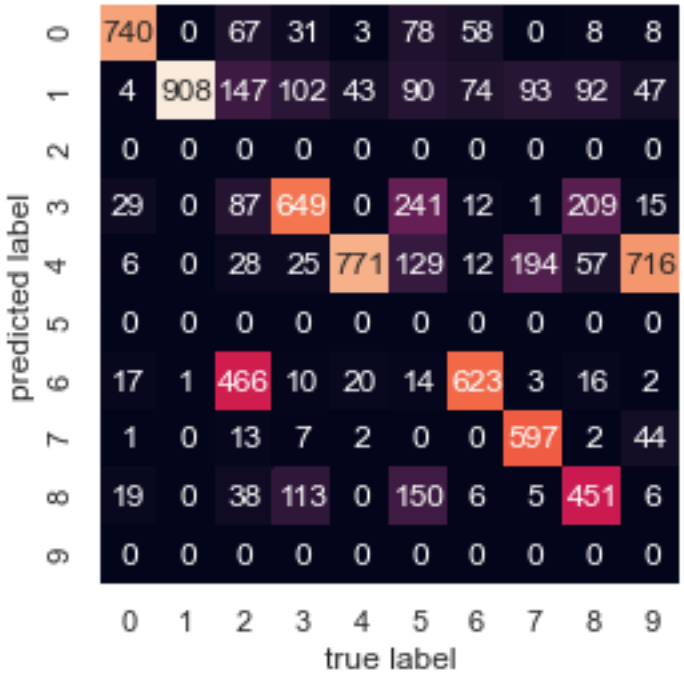
```
array([0, 4, 6, ..., 7, 2, 5])
```

```
y_val_labels = []
for i in range(0,len(y_val_pred)):
    x = y_val_pred[i]
    y_val_labels += [key['Label'].loc[x]]

print(classification_report(y_val, y_val_labels))
```

```
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       816
           1       0.57      1.00      0.72       909
           2       0.00      0.00      0.00       846
           3       0.52      0.69      0.60       937
           4       0.40      0.92      0.56       839
           5       0.00      0.00      0.00       702
           6       0.53      0.79      0.64       785
           7       0.90      0.67      0.77       893
           8       0.57      0.54      0.56       835
           9       0.00      0.00      0.00       838

    accuracy                           0.56      8400
   macro avg       0.42      0.55      0.47      8400
weighted avg       0.43      0.56      0.48      8400
```

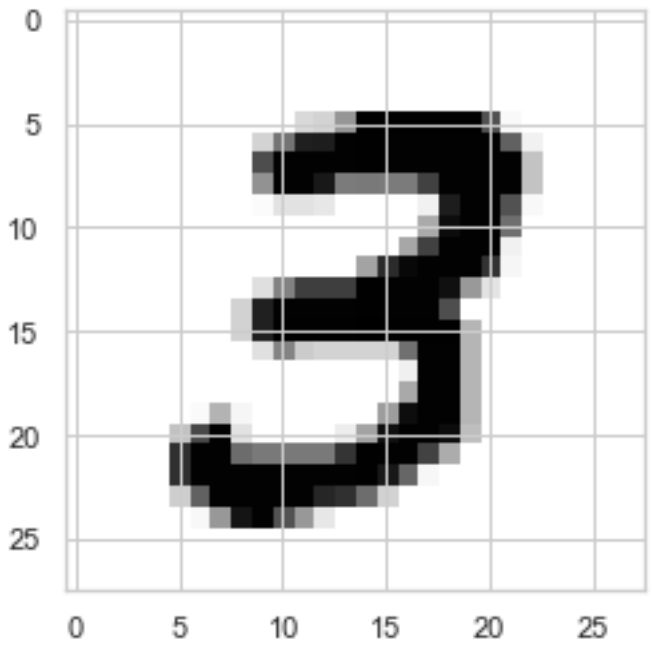
```
Kmeans_pred = kmeans.predict(test)

test_labels = []
for i in range(0,len(Kmeans_pred)):
    x = Kmeans_pred[i]
    test_labels += [key['Label'].loc[x]]

# create submission file
submission = pd.DataFrame({"ImageId": (test.index + 1),"Label":
test_labels})
submission.to_csv('submission_kmeans.csv', index=False)

from IPython.display import Image
Image(filename='submission_kmeans.png')
```

```
mat2 = confusion_matrix(y_val, y_val_labels)
sns.heatmap(mat2.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



ANN

```
# load dataset
from keras.datasets import mnist


# split dataset into training and test set
(x_train, y_train), (x_test, y_test) = mnist.load_data()

plt.imshow(x_train[7], cmap=plt.cm.binary)

<matplotlib.image.AxesImage at 0x1edc84a3b80>
```




```
# View the dimension of tensor

print(x_train.shape)

# View the data type of tensor

print(x_train.dtype)
```

```
(60000, 28, 28)
uint8
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalization
x_train = x_train/255.0
x_test = x_test/255.0

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

print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

```
from tensorflow.keras.utils import to_categorical
```

```
y_train = to_categorical(y_train, num_classes=10)

y_test = to_categorical(y_test, num_classes=10)

print(y_test[0])


print(y_train[0])

print(y_test.shape)
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
(10000, 10)
```

```
# Define the model

model = Sequential()
model.add(Dense(10, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax')) # 2nd layer is a softmax
layer of 10 neurons, which means that it will return a matrix of 10
probability values representing the 10 possible digits.

# the neural network has been defined as a sequence of two layers that
are densely connected (or fully connected)
# all the neurons in each layer are connected to all the neurons in
the next layer

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                7850
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss="categorical_crossentropy",
              optimizer="sgd",
              metrics = ['accuracy'])

model.fit(x_train, y_train, batch_size=100, epochs=10)
```

```
Epoch 1/10
600/600 [==============================] - 2s 3ms/step - loss: 2.1726
```

```
- accuracy: 0.3224
Epoch 2/10
600/600 [==============================] - 2s 3ms/step - loss: 1.9249
- accuracy: 0.5523
Epoch 3/10
600/600 [==============================] - 2s 3ms/step - loss: 1.6993
- accuracy: 0.6453
Epoch 4/10
600/600 [==============================] - 2s 3ms/step - loss: 1.4939
- accuracy: 0.7115
Epoch 5/10
600/600 [==============================] - 2s 3ms/step - loss: 1.3181
- accuracy: 0.7516
Epoch 6/10
600/600 [==============================] - 1s 2ms/step - loss: 1.1740
- accuracy: 0.7768
Epoch 7/10
600/600 [==============================] - 1s 2ms/step - loss: 1.0580
- accuracy: 0.7945
Epoch 8/10
600/600 [==============================] - 1s 2ms/step - loss: 0.9648
- accuracy: 0.8098
Epoch 9/10
600/600 [==============================] - 1s 2ms/step - loss: 0.8893
- accuracy: 0.8216
Epoch 10/10
600/600 [==============================] - 1s 2ms/step - loss: 0.8273
- accuracy: 0.8313

<tensorflow.python.keras.callbacks.History at 0x1edbe56d640>
```

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

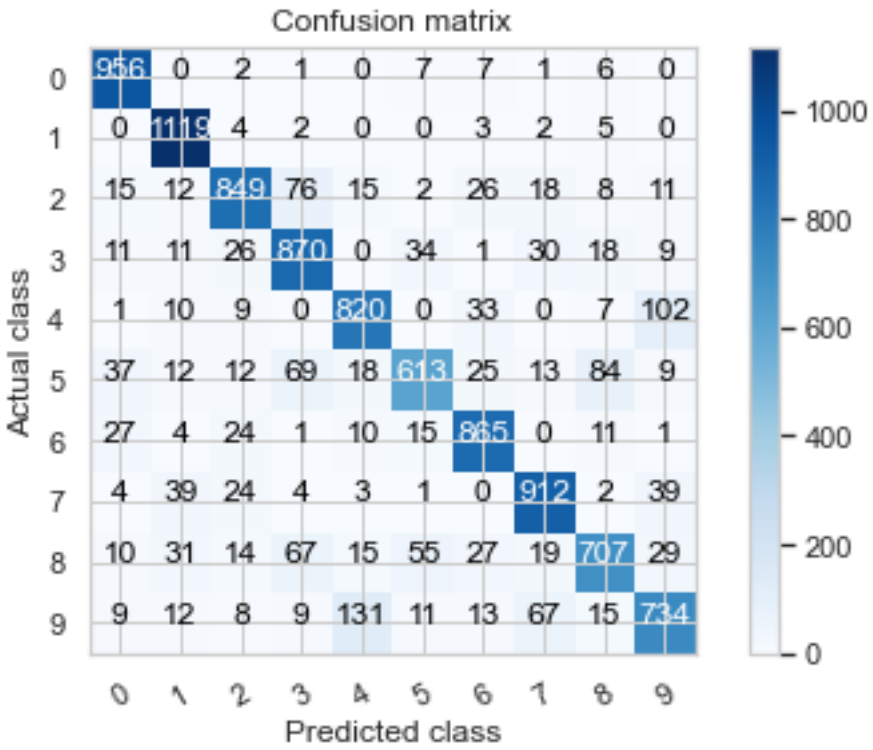
```
313/313 [==============================] - 1s 2ms/step - loss: 0.7847
- accuracy: 0.8445
```

```
print('Test accuracy:', round(test_acc,4))
```

```
Test accuracy: 0.8445
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
```

```
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=30)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Actual class')
    plt.xlabel('Predicted class')

from collections import Counter
import itertools

# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred, axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```




```
test_n = test/255.

predictions = model.predict(test_n)

test_labels = []
for i in range(0,len(predictions)):
    test_labels.append(np.argmax(predictions[i]))

# create submission file
submission = pd.DataFrame({"ImageId": (test_n.index + 1),"Label":
test_labels})
submission.to_csv('NNsubmission.csv', index=False)

from IPython.display import Image
Image(filename='NNsubmission.png')
```

YOUR RECENT SUBMISSION

NNsubmission.csv Score: 0.83782

Submitted by SeafoodTakeout · Submitted just now

Conclusion

The models above ran fairly well. I changed quite a few things in this assignment compared to the last one. Initaily, I was concerned about overfitting the Random Forest model,

however it seemed to work out fine. Additionally, I initially ran my ANN model without applying normalization. It didn't run as well as I expected so I reran it after normalization and it vastly improved the performance.