+ Code + Text

XGBOOST & DNN

Methodology

- **Data Cleaning:** Checking for null values and based on their number either droping them or replacing with mean, median, mode based on the type and description of data. Droping decscrete and catagorical variables that have highly skewed histograms.
- **Data Visualization:** This step helps understand the understand the data in a visually. We can understand normality of the data as well. This helps us to decide whether to normalize the data. In case of catagorical variables it also helps in feature selection.
- **Feature Selection:** Based on the Pearson correlation between the labeled column and rest of the features. In general, a very great correlation should have an absolute value greater than 0.75. When the labeled column is depended on

multiple columns, the correlation with one column may be less. But combined features may have higher effect.

- **Train Test Split:** We split the data into 80:20 ratio for tarining testing respectively.
- **Model Selection:** Based on the data visualization and data correlation, we need to select a model that would best suit. Here we need to use XGBOOST.
- **Evalution:** In this case we are using RMSE, R2 Score to determine the accuracy of

Importing data

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
import numpy as np
import matplotlib.pyplot as plt
%load_ext tensorboard
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
11493376/11490434 [==============================] - 0s 0us/step
```

normalizing the data

```
X_train = X_train/255
X_test = X_test/255
```

checking the shaps of data

```
X_train.shape
```

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

```
X_test.shape
```

```
(10000, 28, 28)
```

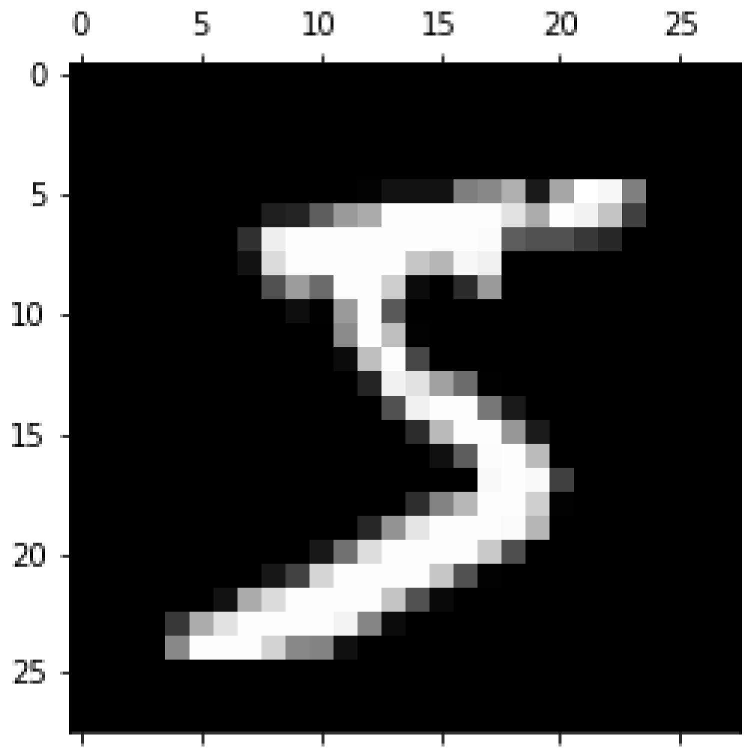
one hot encoding of label feature

```
import keras
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

ploting data

```
plt.gray()
plt.matshow(X_train[0])
np.argmax(y_train[0])
```

```
5
<Figure size 432x288 with 0 Axes>
```



creating a simple NN on the data

```
import tensorflow as tf
from keras.layers import Dense
```

```
from keras.models import Sequential
```

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation="relu"),
                                    tf.keras.layers.Dense(10, activation="softmax"
```

training the NN on the data

```
tf.keras.backend.clear_session()
model.compile(optimizer="Adam", loss='categorical_crossentropy', metrics=['accurac
history = model.fit(X_train, y_train, batch_size=128, epochs=50, validation_split=
```

```
Epoch 1/50
  1/422 [..............................] - ETA: 0s - loss: 2.3343 - accuracy
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
  2/422 [..............................] - ETA: 18s - loss: 2.3052 - accurac
422/422 [==============================] - 2s 4ms/step - loss: 0.3814 - accu
Epoch 2/50
422/422 [==============================] - 2s 4ms/step - loss: 0.1817 - accu
Epoch 3/50
422/422 [==============================] - 2s 4ms/step - loss: 0.1303 - accu
Epoch 4/50
422/422 [==============================] - 2s 4ms/step - loss: 0.1016 - accu
Epoch 5/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0821 - accu
Epoch 6/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0668 - accu
Epoch 7/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0557 - accu
Epoch 8/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0475 - accu
Epoch 9/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0393 - accu
Epoch 10/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0338 - accu
Epoch 11/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0286 - accu
Epoch 12/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0245 - accu
Epoch 13/50
```

```
Epoch 13/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0204 - accu
Epoch 14/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0169 - accu
Epoch 15/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0149 - accu
Epoch 16/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0119 - accu
Epoch 17/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0101 - accu
Epoch 18/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0098 - accu
Epoch 19/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 - accu
Epoch 20/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0067 - accu
Epoch 21/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0054 - accu
Epoch 22/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0042 - accu
Epoch 23/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0040 - accu
Epoch 24/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0038 - accu
Epoch 25/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0061 - accu
Epoch 26/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0049 - accu
Epoch 27/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0020 - accu
```

```
422/422 [==============================] - 2s 4ms/step - loss: 0.0026 - accu
Epoch 28/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0013 - accu
Epoch 29/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0013 - accu
Epoch 30/50
422/422 [==============================] - 2s 4ms/step - loss: 8.9793e-04 -
Epoch 31/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0024 - accu
Epoch 32/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0105 - accu
Epoch 33/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0017 - accu
Epoch 34/50
422/422 [==============================] - 2s 4ms/step - loss: 7.0396e-04 -
Epoch 35/50
422/422 [==============================] - 2s 4ms/step - loss: 5.3370e-04 -
Epoch 36/50
422/422 [==============================] - 2s 4ms/step - loss: 4.4426e-04 -
Epoch 37/50
422/422 [==============================] - 2s 4ms/step - loss: 3.9125e-04 -
Epoch 38/50
422/422 [==============================] - 2s 4ms/step - loss: 3.6494e-04 -
Epoch 39/50
422/422 [==============================] - 2s 4ms/step - loss: 4.8519e-04 -
Epoch 40/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0142 - accu
Epoch 41/50
422/422 [==============================] - 2s 4ms/step - loss: 0.0017 - accu
Epoch 42/50
```

```
Epoch 42/50
422/422 [==============================] - 2s 4ms/step - loss: 7.9262e-04 -
Epoch 43/50
422/422 [==============================] - 2s 4ms/step - loss: 3.6454e-04 -
Epoch 44/50
422/422 [==============================] - 2s 4ms/step - loss: 2.9782e-04 -
Epoch 45/50
```

Testing the NN on the data

```
Epoch 47/50
```

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1019 - accu
[0.1018671840429306, 0.9790999889373779]
```

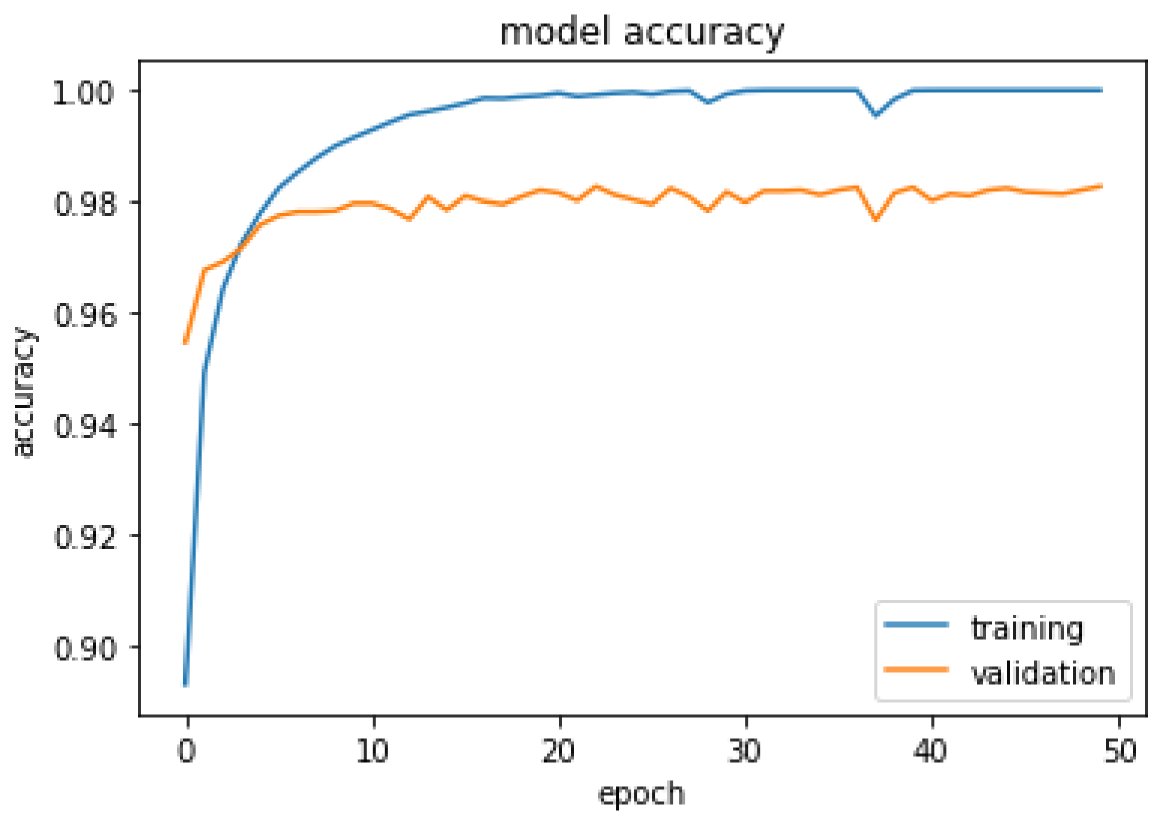
```
Epoch 50/50
```

there is no overfit in the model

ploting training and validation accuracy

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



creating heat map for confusion matrix

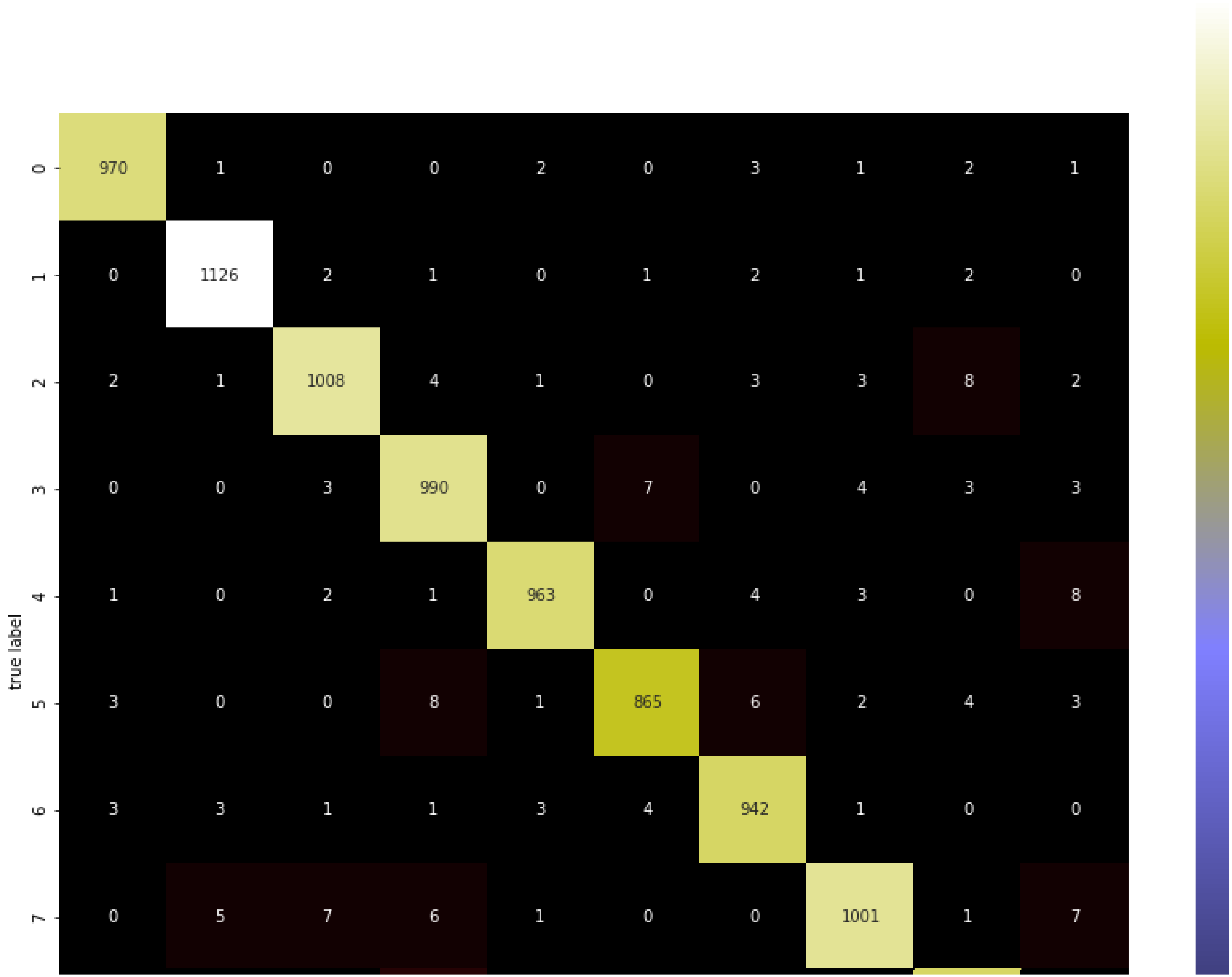
```
import numpy as np
p=model.predict(X_test)
pred=[]
act=[]
```

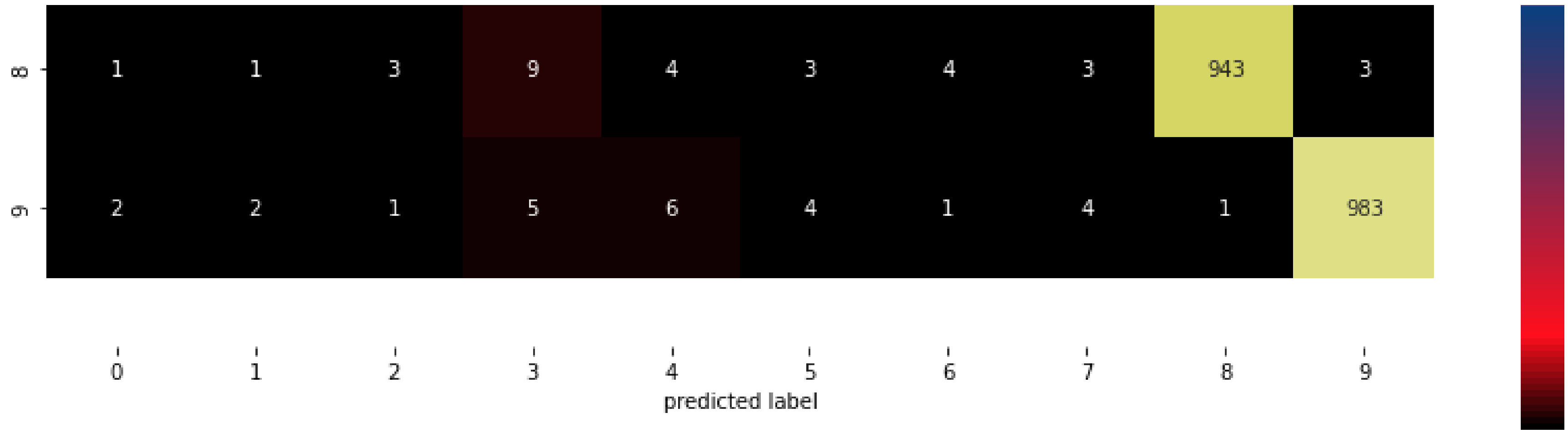
```
for i in range(len(X_test)):
    act.append(np.argmax(y_test[i]))
    pred.append(np.argmax(p[i]))
```

```
from sklearn.metrics import confusion_matrix
confm=confusion_matrix(act, pred, labels=list(range(10)))
```

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(15,15))
ax=sns.heatmap(confm,annot=True,fmt='d',cbar=True,square=True,cmap="gist_stern")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.xlabel("predicted label")
plt.ylabel("true label")
```

```
Text(114.0, 0.5, 'true label')
```





The errors are spread out in all the classes

training XGBoost on Data

```
from xgboost import XGBClassifier
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train=X_train.reshape(60000,784) # flattening the data
X_test=X_test.reshape(10000,784)
model = XGBClassifier()
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
model.score(X_train, y_train)
```

```
0.9434833333333333
```

Testing XGBoost on the data

There no overfit

```
model.score(X_test, y_test)
```

```
0.9368
```

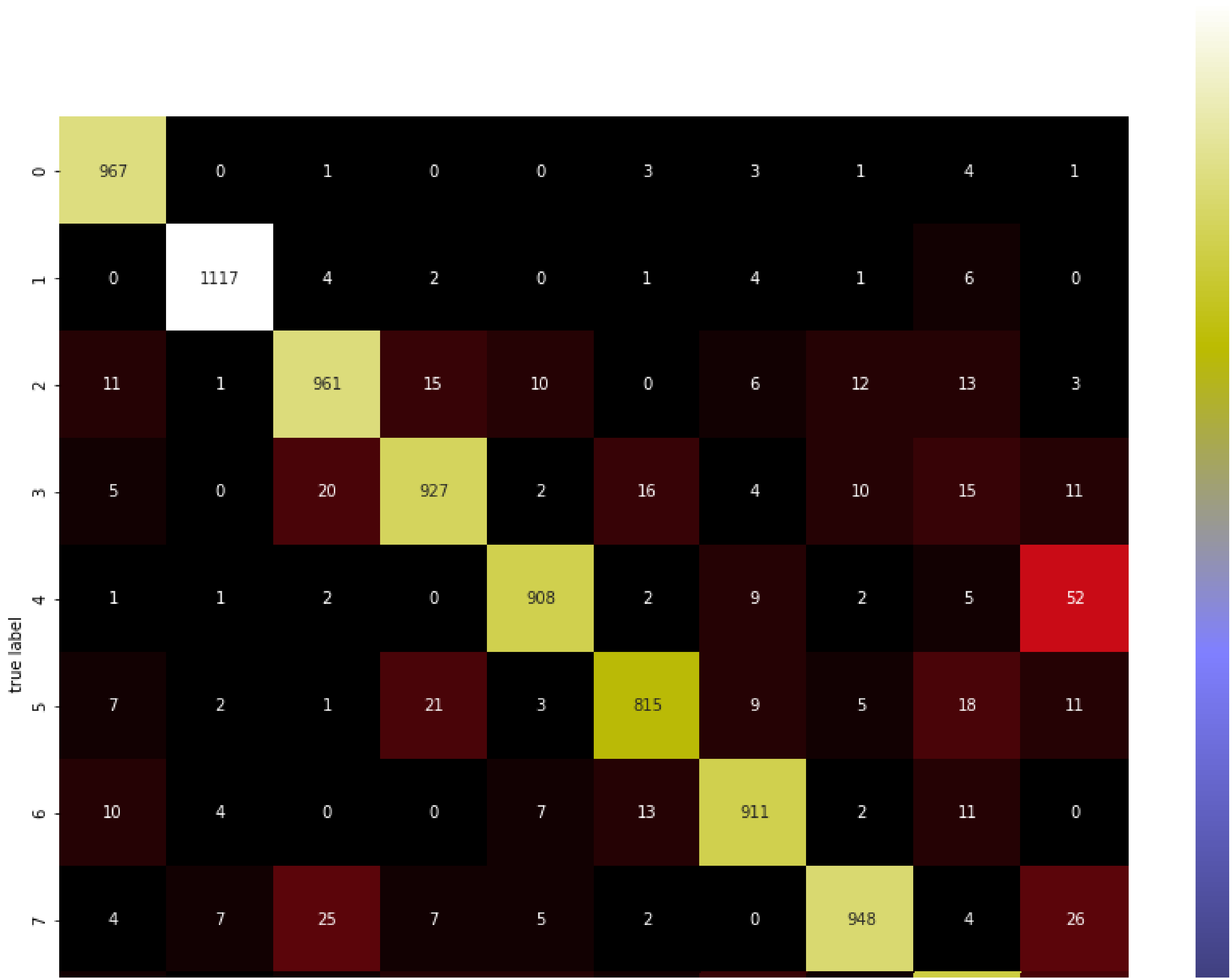
```
import numpy as np
p=model.predict(X_test)
```

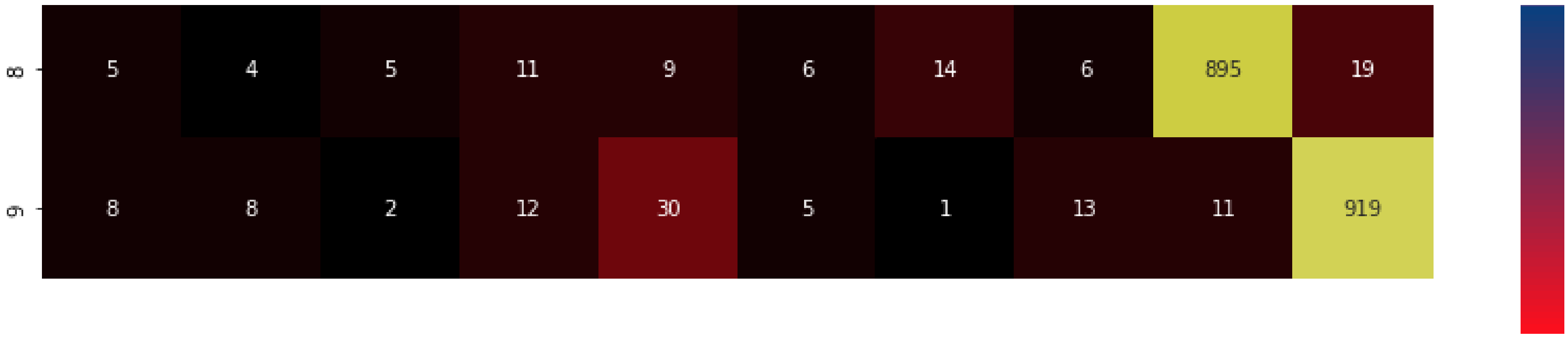
```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import confusion_matrix
confm=confusion_matrix(y_test, p, labels=list(range(10)))
```

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(15,15))
ax=sns.heatmap(confm,annot=True,fmt='d',cbar=True,square=True,cmap="gist_stern")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.xlabel("predicted label")
plt.ylabel("true label")
```

```
Text(114.0, 0.5, 'true label')
```





The errors are not so speardout and are more in number compaed to NN

Class 4 has 52 intsances clasified as 9 there is scope for learning