Beverlyn Tucker

IST707 Data Mining Analysis Hand Written Digits HW6&7

Introduction

Handwriting recognition provides various organizations with the ability to convert text documents into a digitized equivalent.

The dataset of 60,000 training set and 10,000 test sets composed of small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9

In these generations, there are many algorithms exist to perform classification techniques. However, not all are appropriate for the task at hand. Additionally, choosing the most appropriate algorithm is often not a trivial task. Various factors, including accuracy and computational time, are factors requiring careful consideration. In data science, concepts of aggregating data, visualization, and generation of models for predictive and prescriptive analytics are major areas of focus.

Having prepackaged libraries in python or R, consolidates code, allowing functions to be easily implemented. Since machine learning algorithms can quickly get complicated with calculations and ensembling, sometimes the easiest way to determine performance is to determine the runtime of a function. This can be done by comparing the runtime of multiple functions to pass the same dataset(s). Therefore, multiple techniques, such as classification algorithms, can be compared for accuracy, as well as runtime performance.

In this study, various models will be fitted using decision trees, Naïve Bayes, support vector machines (svm), k-nearest neighbors (knn), and random forests. Specifically, these classification algorithms will attempt to predict numeric values, represented as a series of pixels, where numeric values indicate color darkness. Prediction accuracy will be compared between the three algorithms, as well as runtime performance. These results will then be compared to previous findings, regarding predicting numerical text, with respect to naïve Bayes and decision trees.

```
from keras.layers import Dense, Conv2D, Flatten
from tensorflow.keras import layers               # for Neural Network layers
from tensorflow.keras import datasets             # for keras MNIST datasets
from sklearn.metrics import confusion_matrix      # for prediction verification
import matplotlib.pyplot as plt                   # for plotting
import numpy as np                                # for np functions
import seaborn as sns; sns.set()                  # for heatmap
from sklearn.metrics import classification_report # to print classification accuracy
from tensorflow.keras.utils import to_categorical # to convert data to categorical
from tensorflow.keras import layers               # for Neural Network layers
from tensorflow.keras import Sequential           # for Neural Network
from keras.callbacks import ModelCheckpoint       #for  Neural Network callback
from sklearn.naive_bayes import GaussianNB        # for GNB
from sklearn.tree import DecisionTreeClassifier   # for decision Tree classfier
from keras.models import Model                    #for Keras Model
from sklearn import svm                           # for svm
from sklearn.neighbors import KNeighborsClassifier # knn
from sklearn.ensemble import RandomForestClassifier# randomforest
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold
```

```
Using TensorFlow backend.
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use th
  import pandas.util.testing as tm
```

Data Preparation:

Data from mnist datasets representing instances of numeric digits ranging from 0-9 are used. More generally, each numeric digit was represented by a sequence of (roughly 780) pixels. Each pixel contains a numeric value associated with the pixel density. Then reshape the data to 20 28 by 28. The created a label in each number.

Importing Dataset

```
##loading data from tensorflow.keras import datasets
(x_train, y_train), (x_test, y_test) = (datasets.mnist.load_data())
```

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

```
# Split the train and the validation set for the fitting

np.random.seed(1234)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.1)
```

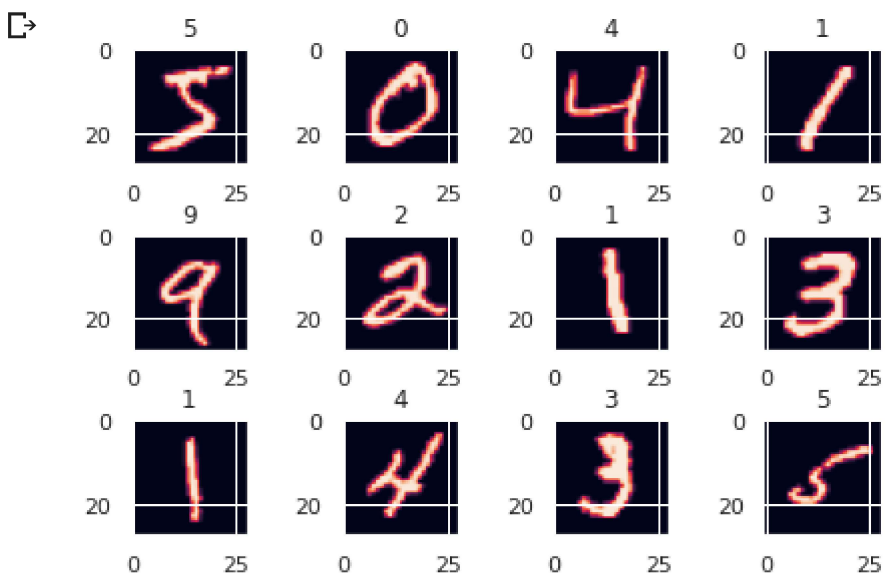
Image labels for the target images

```
label_name = [ '0', '1','2','3','4','5','6','7','8','9']


x_train = x_train.astype('float32')
y_train = y_train.astype('int32')

x_test = x_test.astype('float32')
y_test = y_test.astype('int32')

##x_val =x_val.astype('int32')
##y_val =y_val.astype('int32')

# Normalize the color values to 0-1
# (as imported, they're 0-255)
x_train /= 255
x_test /= 255

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

Train labels dataset

```
fig, ax = plt.subplots(3, 4)
fig.tight_layout()
for i, axi in enumerate(ax.flat):
  axi.imshow(x_train[i])
  axi.set_title(label_name[y_train[i]])
plt.show()
```



GNB_Model Reshape input data to naive bayes algorithm

```
from sklearn.naive_bayes import GaussianNB        # for GNB
w, h = 28, 28
gnb_x_train = x_train.reshape(x_train.shape[0], w*h)
gnb_x_test = x_test.reshape(x_test.shape[0], w*h)

gnb_model = GaussianNB()
results_skfold = (model_selection.cross_val_score(model_skfold, x_train, y_train, cv=skfold))


%time gnb_model.fit(gnb_x_train, y_train)
```

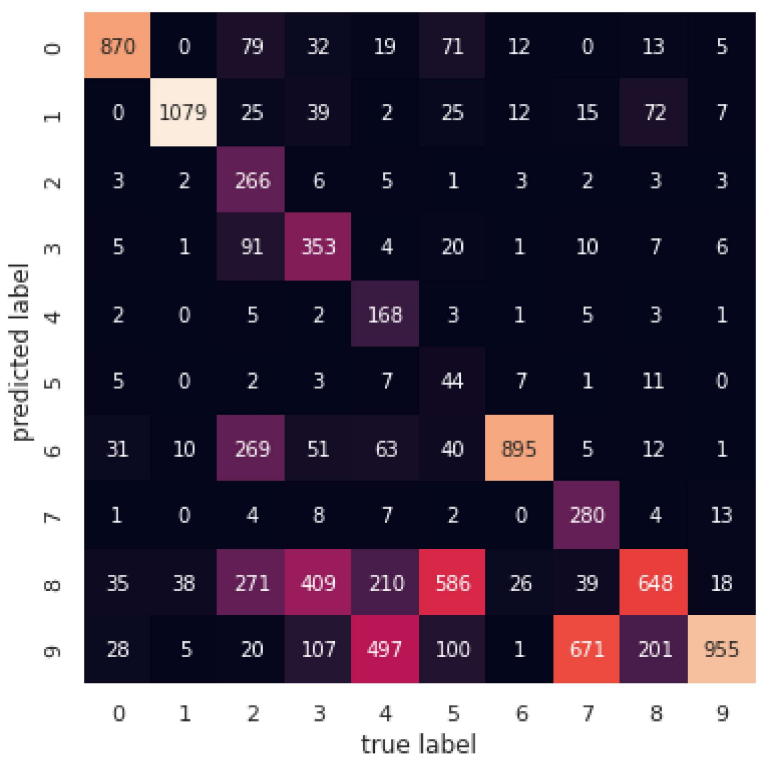
```
CPU times: user 331 ms, sys: 5.79 ms, total: 337 ms
Wall time: 340 ms
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
gnb_trn_fit = gnb_model.predict(gnb_x_test )
print(classification_report(y_test, gnb_trn_fit, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       980
           1       0.85      0.95      0.90      1135
           2       0.90      0.26      0.40      1032
           3       0.71      0.35      0.47      1010
           4       0.88      0.17      0.29       982
           5       0.55      0.05      0.09       892
           6       0.65      0.93      0.77       958
           7       0.88      0.27      0.42      1028
           8       0.28      0.67      0.40       974
           9       0.37      0.95      0.53      1009

    accuracy                           0.56     10000
   macro avg       0.69      0.55      0.51     10000
weighted avg       0.69      0.56      0.52     10000
```

GNB Confusion_matrix

```
mat = confusion_matrix(y_test, gnb_trn_fit)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



DecisionTree Classifier

Reshape input data to fit decision tree algorithm splitter='best',min_samples_split=3

```
w, h = 28, 28
dtc_x_train = x_train.reshape(x_train.shape[0], w*h)
dtc_x_test = x_test.reshape(x_test.shape[0], w*h)
```

```
dtc_model = DecisionTreeClassifier(criterion="gini",splitter='best',min_samples_split=3)
%time dtc_model.fit(dtc_x_train, y_train)
dtc_model2 = DecisionTreeClassifier(criterion="entropy",splitter='best',min_samples_split=3)
%time dtc_model.fit(dtc_x_train, y_train)
```

```
CPU times: user 21.7 s, sys: 7.99 ms, total: 21.7 s
Wall time: 21.7 s
CPU times: user 22 s, sys: 2.99 ms, total: 22 s
Wall time: 22 s
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=3,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

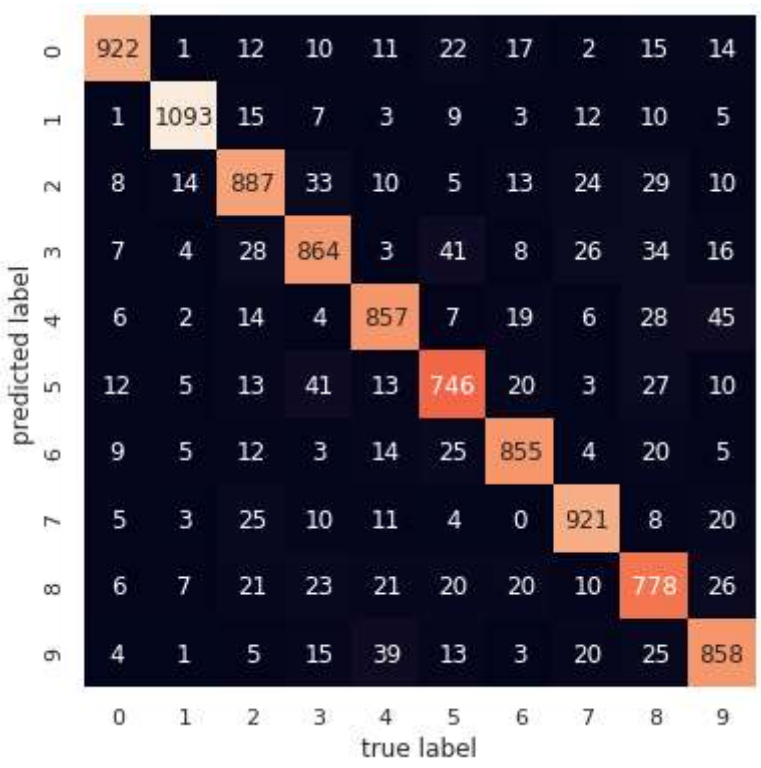
```
dtc_y_fit = dtc_model.predict(dtc_x_test)
print(classification_report(y_test, dtc_y_fit, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       980
           1       0.95      0.96      0.95      1135
           2       0.86      0.86      0.86      1032
           3       0.83      0.84      0.84      1010
           4       0.88      0.87      0.88       982
           5       0.85      0.84      0.84       892
           6       0.90      0.88      0.89       958
           7       0.92      0.89      0.91      1028
           8       0.83      0.82      0.82       974
           9       0.85      0.85      0.85      1009

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

```
w, h = 28, 28
dtc_x_train = x_train.reshape(x_train.shape[0], w*h)
dtc_x_test = x_test.reshape(x_test.shape[0], w*h)

dtc_model2 = DecisionTreeClassifier(criterion="entropy",splitter='best',min_samples_split=3)
%time dtc_model2.fit(dtc_x_train, y_train)

dtc_y_fit = dtc_model2.predict(dtc_x_test)
print(classification_report(y_test, dtc_y_fit, target_names=label_name))
```

```
CPU times: user 22 s, sys: 998 µs, total: 22 s
Wall time: 22 s
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       980
           1       0.97      0.97      0.97      1135
           2       0.87      0.89      0.88      1032
           3       0.85      0.84      0.85      1010
           4       0.87      0.87      0.87       982
           5       0.82      0.85      0.84       892
           6       0.90      0.89      0.90       958
           7       0.92      0.90      0.91      1028
           8       0.85      0.83      0.84       974
           9       0.87      0.86      0.86      1009

    accuracy                           0.89     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.89      0.89      0.89     10000
```

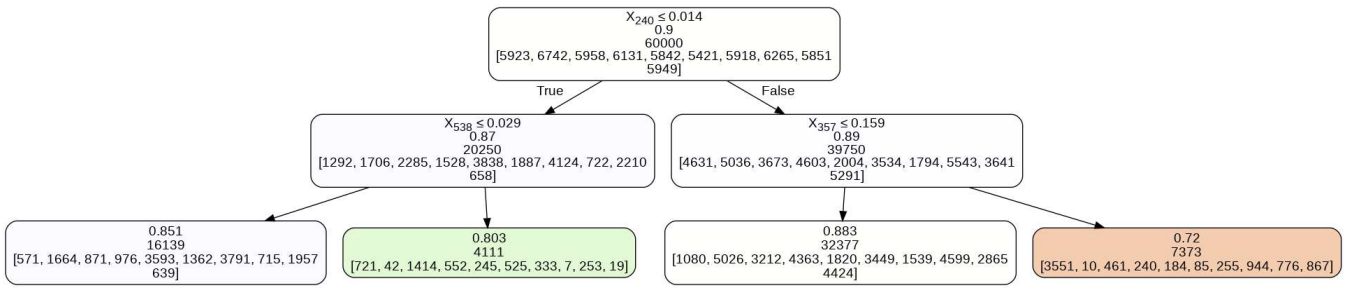
```
mat = confusion_matrix(y_test, dtc_y_fit)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Tree Criterion Gini max depth 2 max feature 2 with 3 split

```
##Tree required Packages
from sklearn import tree
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
w, h = 28, 28
tree_x_train = x_train.reshape(x_train.shape[0], w*h)
tree_x_test = x_test.reshape(x_test.shape[0], w*h)



clf = tree.DecisionTreeClassifier(criterion="gini", max_depth=2, max_features=2,min_samples_split=3)
clf = clf.fit(tree_x_train, y_train)
%time clf.fit(tree_x_train, y_train)

dot_data = StringIO()
export_graphviz(clf, out_file =dot_data, filled=True, rounded =True, special_characters=True, label='label_name')
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

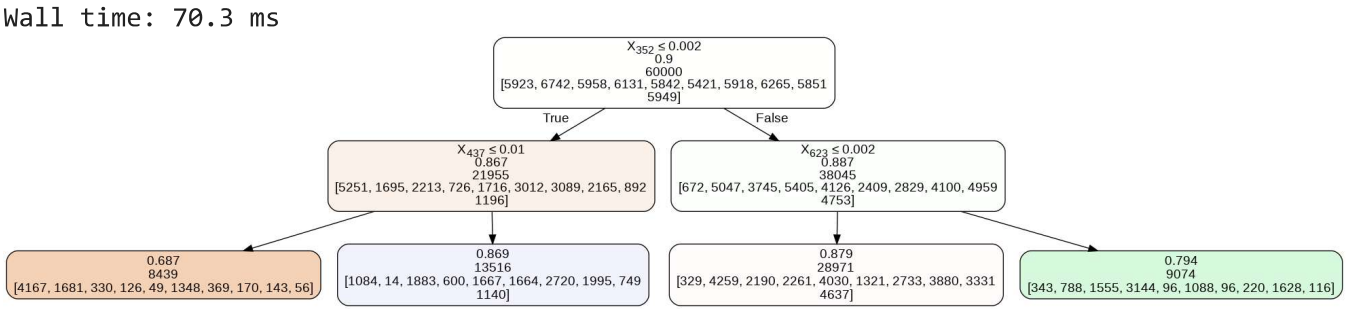
```
CPU times: user 65 ms, sys: 0 ns, total: 65 ms
Wall time: 65.1 ms
```



Criterion Entropy max depth 2 and maz feature 2 with 3 split

```
w, h = 28, 28
tree_x_train = x_train.reshape(x_train.shape[0], w*h)
tree_x_test = x_test.reshape(x_test.shape[0], w*h)

clf2 = tree.DecisionTreeClassifier(criterion="entropy", max_depth=2, max_features=2, min_samples_split=3)
clf2 = clf2.fit(tree_x_train, y_train)
%time clf2.fit(tree_x_train, y_train)

dot_data = StringIO()
export_graphviz(clf, out_file =dot_data, filled=True, rounded =True, special_characters=True, label='label_name')
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
CPU times: user 69.9 ms, sys: 0 ns, total: 69.9 ms
Wall time: 70.3 ms
```



SVM Model Kernel =rbf

```
w, h = 28, 28
rbf_x_train = x_train.reshape(x_train.shape[0], w*h)
rbf_x_test = x_test.reshape(x_test.shape[0], w*h)

rbf_model = svm.SVC(C=10000.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
  gamma=0.001, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False)
%time rbf_model.fit(rbf_x_train, y_train)
```

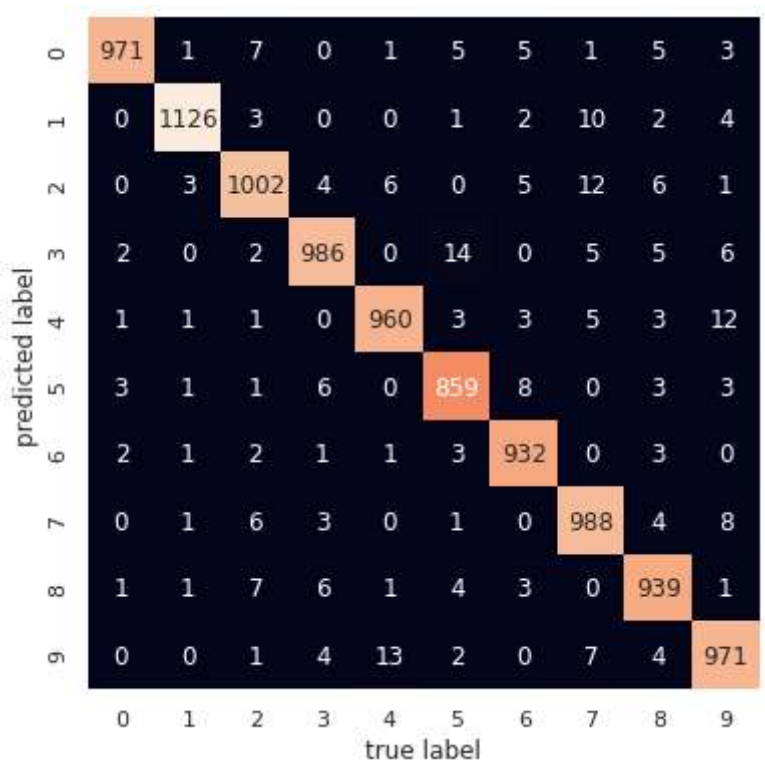
```
CPU times: user 6min 57s, sys: 184 ms, total: 6min 58s
Wall time: 6min 58s
SVC(C=10000.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
rbf_y_fit = rbf_model.predict(rbf_x_test )
print(classification_report(y_test, rbf_y_fit, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.96      0.97      0.97      1032
           3       0.97      0.98      0.97      1010
           4       0.97      0.98      0.97       982
           5       0.97      0.96      0.97       892
           6       0.99      0.97      0.98       958
           7       0.98      0.96      0.97      1028
           8       0.98      0.96      0.97       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

SVM rbf Confusion Matrix

```
mat = confusion_matrix(y_test, rbf_y_fit)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



SMV rbf Results

the numbers that misclassify are seven predicted as 9 and as 2 both 11 times each, and four predicted as 9. Time wise this took 935 ms with over aall accurary of 97%

SVM Model Kernel = Linear

```
w, h = 28, 28
linear_x_train = x_train.reshape(x_train.shape[0], w*h)
linear_x_test = x_test.reshape(x_test.shape[0], w*h)

linear_model = svm.SVC(kernel='linear', C=0.10)
%time linear_model.fit(linear_x_train , y_train)
```

```
CPU times: user 7min 26s, sys: 303 ms, total: 7min 26s
Wall time: 7min 27s
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

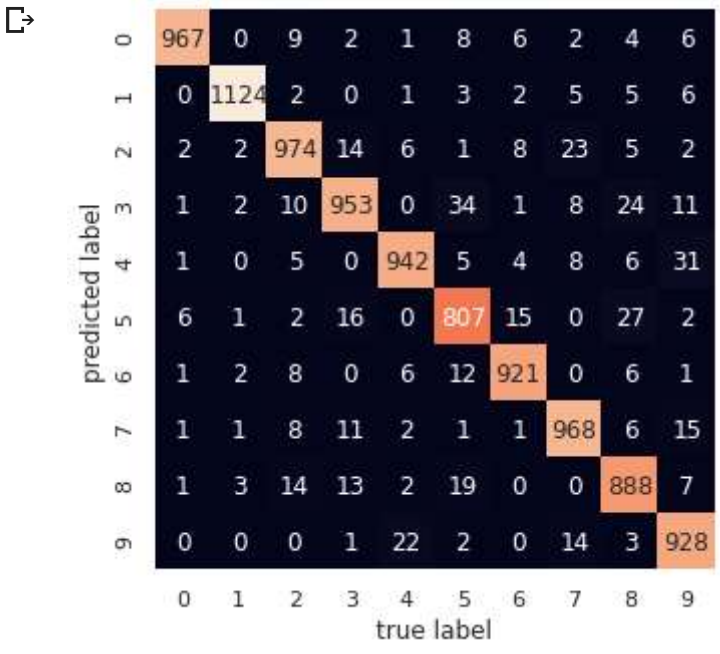
SVM Linear Prediction Results

```
linear_y_fit = linear_model.predict(linear_x_test )
print(classification_report(y_test, linear_y_fit, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       980
           1       0.98      0.99      0.98      1135
           2       0.94      0.94      0.94      1032
           3       0.91      0.94      0.93      1010
           4       0.94      0.96      0.95       982
           5       0.92      0.90      0.91       892
           6       0.96      0.96      0.96       958
           7       0.95      0.94      0.95      1028
           8       0.94      0.91      0.92       974
           9       0.96      0.92      0.94      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

SVM Linear Confusion Matrix

```
mat = confusion_matrix(y_test, linear_y_fit)
plt.figure(figsize=(5, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



SVM on Linear Results

Number with the highest misclassification are Five miss-predicted as number three for 34 times and was 9 for 19 in a row. And true label eight predicted as number 5 for 27 times and as 3 for 24 times.

Overall on SVM model, Kernel rbf is better with total accuracy of 97% compared to 95% accuracy on linear. Time-wise rbf it took a little bit long than linear.

knn Model 1 with KNeighbors 5

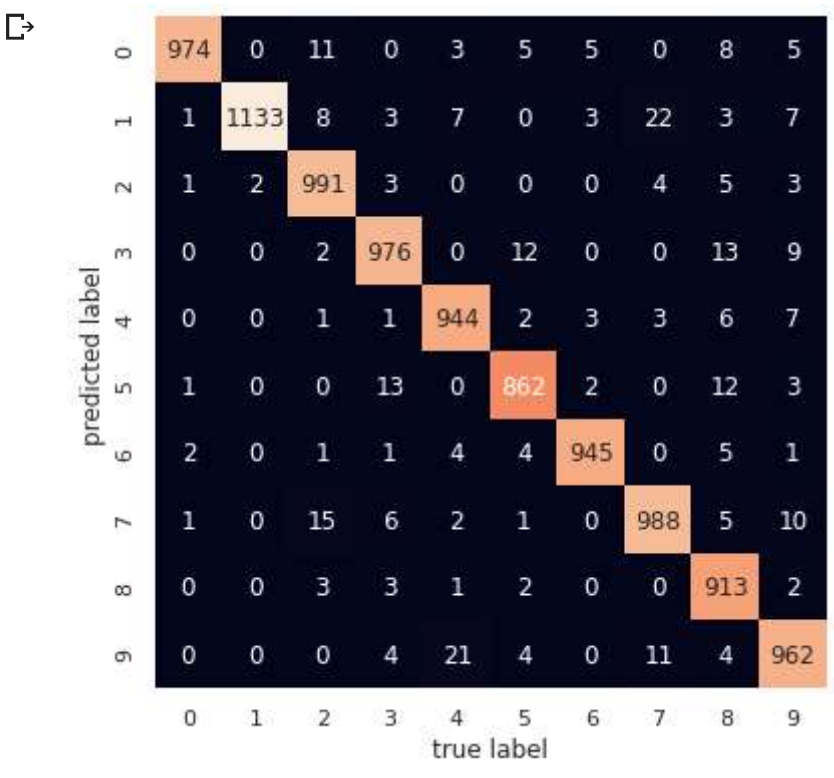
```
w, h = 28, 28
knn_x_train = x_train.reshape(x_train.shape[0], w*h)
knn_x_test = x_test.reshape(x_test.shape[0], w*h)

knn_model = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
%time knn_model.fit(knn_x_train, y_train)
knn_y_fit = knn_model.predict(knn_x_test )
print(classification_report(y_test, knn_y_fit, target_names=label_name))
```

```
CPU times: user 15.9 s, sys: 177 ms, total: 16.1 s
Wall time: 16.1 s
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       980
           1       0.95      1.00      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.97      0.97      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

knn Model Prediction Results with 5 Kneighbors

knn Confusion Matrix

```
mat = confusion_matrix(y_test, knn_y_fit)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



knn with 5 KNeighbors

Number Four missclassify as number 9 and seven as a number 1

Result

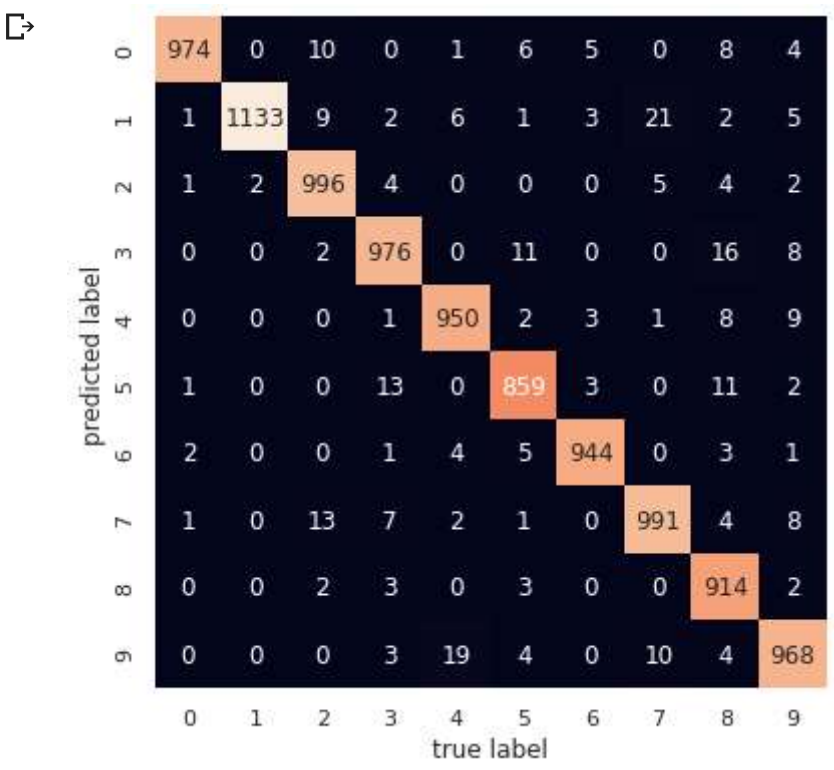
The number that has a higher misclassification are number nine mistakenly predict as 4, followed by number one as 7

knn Model 2 with KNeighbors 3

```
w, h = 28, 28
knn_x_train2 = x_train.reshape(x_train.shape[0], w*h)
knn_x_test2 = x_test.reshape(x_test.shape[0], w*h)

knn_model2 = KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=2)
%time knn_model2.fit(knn_x_train2, y_train)

knn_y_fit2 = knn_model2.predict(knn_x_test2)
print(classification_report(y_test, knn_y_fit2, target_names=label_name))
```

```
CPU times: user 15.9 s, sys: 184 ms, total: 16 s
Wall time: 16 s
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.96      1.00      0.98      1135
           2       0.98      0.97      0.97      1032
           3       0.96      0.97      0.96      1010
           4       0.98      0.97      0.97       982
           5       0.97      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

```
mat = confusion_matrix(y_test, knn_y_fit2)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Result on knn Model

KNieghbors =5 and 3 at this time has sligh variation only two and three different from previously misclassification

overall on knn have the same accuracy of 97% and the time only 1 sec. difference

Random Forest Model

Random Forest Model Prediction Criterion entropy

```
w, h = 28, 28
forest_x_train = x_train.reshape(x_train.shape[0], w*h)
forest_x_test = x_test.reshape(x_test.shape[0], w*h)

forest_model = RandomForestClassifier(n_estimators=10, criterion='entropy',random_state=0)
%time forest_model.fit(forest_x_train, y_train)
```

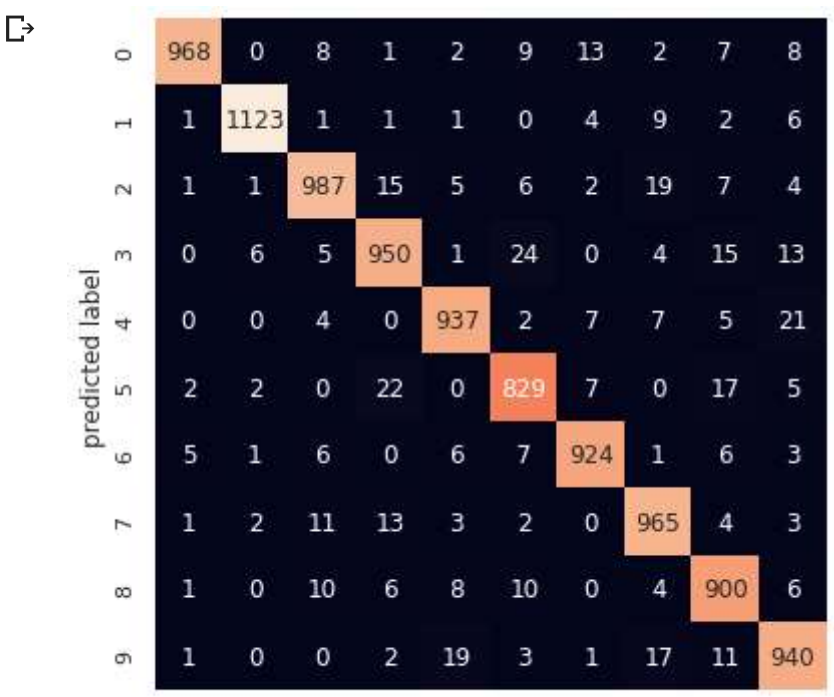
```
CPU times: user 6.94 s, sys: 6.97 ms, total: 6.95 s
Wall time: 6.95 s
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
```

```
forest_y_fit = forest_model.predict(forest_x_test )
print(classification_report(y_test, forest_y_fit, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       980
           1       0.98      0.99      0.98      1135
           2       0.94      0.96      0.95      1032
           3       0.93      0.94      0.94      1010
           4       0.95      0.95      0.95       982
           5       0.94      0.93      0.93       892
           6       0.96      0.96      0.96       958
           7       0.96      0.94      0.95      1028
           8       0.95      0.92      0.94       974
           9       0.95      0.93      0.94      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

Random Forest Model Confusion Matrix

```
mat = confusion_matrix(y_test, forest_y_fit)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



RandomForest criterion entropy with estimator equal to 10 The Number with highest misclassification are Five predicted as 3 for (24x), Four predicted as number 9 for 19 times, and nine predicted as 4 (21x) and 3(13x). With 95% accurary and run time of 744 ms

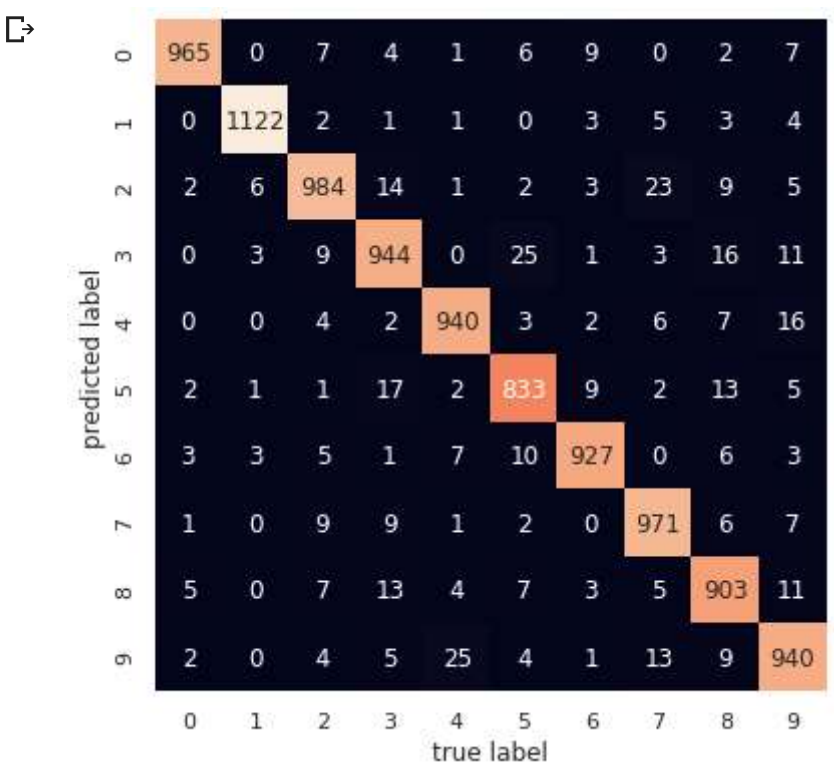
```
w, h = 28, 28
forest_x_train_g = x_train.reshape(x_train.shape[0], w*h)
forest_x_test_g = x_test.reshape(x_test.shape[0], w*h)

forest_model_g = RandomForestClassifier(n_estimators=10, criterion='gini',
                 random_state=0, oob_score=True,min_samples_split=3)
%time forest_model_g.fit(forest_x_train_g, y_train)
```

```
CPU times: user 5.48 s, sys: 3.98 ms, total: 5.49 s


  warn("Some inputs do not have OOB scores. "
```

```
forest_y_fit_g = forest_model_g.predict(forest_x_test_g )
print(classification_report(y_test, forest_y_fit_g, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.98      0.99      0.99      1135
           2       0.94      0.95      0.95      1032
           3       0.93      0.93      0.93      1010
           4       0.96      0.96      0.96       982
           5       0.94      0.93      0.94       892
           6       0.96      0.97      0.96       958
           7       0.97      0.94      0.95      1028
           8       0.94      0.93      0.93       974
           9       0.94      0.93      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

```
mat = confusion_matrix(y_test, forest_y_fit_g)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



RandomForest criterion gini Number with true label Five predicted as 3 for 34 times, seven predicted as 2 for 27 times and nine predicted as 4(29x) and 3(11x).

Overall the model with the same number of estemitor and criterion entropy and gini with the same accuracy of 95%

Additional Neural Network

I decided to do a Neural Network Model due to the curiosity that the neural network is great on classification. I wonder how it does on written digits.

```
##Train the model
w, h = 28, 28
nnt_x_train = x_train.reshape(x_train.shape[0], w, h, 1)
nnt_x_test = x_test.reshape(x_test.shape[0], w, h, 1)
nnt_y_train = to_categorical(y_train, 10)

nnt_model = Sequential()

# Must define the input shape in the first layer of the neural network
nnt_model.add(layers.Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28,28,1)))
nnt_model.add(layers.MaxPooling2D(pool_size=2))
nnt_model.add(layers.Dropout(0.3))

nnt_model.add(layers.Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
nnt_model.add(layers.MaxPooling2D(pool_size=2))
nnt_model.add(layers.Dropout(0.3))

nnt_model.add(layers.Flatten())
nnt_model.add(layers.Dense(256, activation='relu'))
nnt_model.add(layers.Dropout(0.5))
nnt_model.add(layers.Dense(10, activation='softmax'))
```

```
nnt_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Neural Network Epoch = 10

```
## add validation split to avid over fitting
hist= nnt_model.fit(nnt_x_train, nnt_y_train, batch_size=64, epochs=10, validation_split =0.2)
```

```
Epoch 1/10
750/750 [==============================] - 61s 82ms/step - loss: 0.2689 - accuracy: 0.9174 - val_loss: 0.1015 - val_accuracy: 0.9
Epoch 2/10
750/750 [==============================] - 60s 81ms/step - loss: 0.1125 - accuracy: 0.9658 - val_loss: 0.0655 - val_accuracy: 0.9
Epoch 3/10
750/750 [==============================] - 61s 81ms/step - loss: 0.0855 - accuracy: 0.9735 - val_loss: 0.0588 - val_accuracy: 0.9
Epoch 4/10
750/750 [==============================] - 65s 87ms/step - loss: 0.0688 - accuracy: 0.9782 - val_loss: 0.0523 - val_accuracy: 0.9
Epoch 5/10
750/750 [==============================] - 61s 82ms/step - loss: 0.0580 - accuracy: 0.9822 - val_loss: 0.0528 - val_accuracy: 0.9
Epoch 6/10
750/750 [==============================] - 61s 82ms/step - loss: 0.0508 - accuracy: 0.9840 - val_loss: 0.0504 - val_accuracy: 0.9
Epoch 7/10
750/750 [==============================] - 61s 81ms/step - loss: 0.0454 - accuracy: 0.9848 - val_loss: 0.0499 - val_accuracy: 0.9
Epoch 8/10
750/750 [==============================] - 60s 80ms/step - loss: 0.0383 - accuracy: 0.9876 - val_loss: 0.0494 - val_accuracy: 0.9
Epoch 9/10
750/750 [==============================] - 61s 82ms/step - loss: 0.0369 - accuracy: 0.9875 - val_loss: 0.0507 - val_accuracy: 0.9
Epoch 10/10
750/750 [==============================] - 61s 82ms/step - loss: 0.0329 - accuracy: 0.9885 - val_loss: 0.0473 - val_accuracy: 0.9
```

Result copied below got cut off the validation part

Epoch 1/10 750/750 [==] - 61s 82ms/step - loss: 0.2689 - accuracy: 0.9174 - val_loss: 0.1015 - val_accuracy: 0.9718 Epoch 2/10750/750 [==] - 60s 81ms/step - loss: 0.1125 - accuracy: 0.9658 - val_loss: 0.0655 - val_accuracy: 0.9812 Epoch 3/10 750/750 [==] - 61s 81ms/step - loss: 0.0855 - accuracy: 0.9735 - val_loss: 0.0588 - val_accuracy: 0.9826 Epoch 4/10 750/750 [==] - 65s 87ms/step - loss: 0.0688 - accuracy: 0.9782 - val_loss: 0.0523 - val_accuracy: 0.9844 Epoch 5/10 750/750 [==] - 61s 82ms/step - loss: 0.0580 - accuracy: 0.9822 - val_loss: 0.0528 - val_accuracy: 0.9854 Epoch 6/10 750/750 [==] - 61s 82ms/step - loss: 0.0508 - accuracy: 0.9840 - val_loss: 0.0504 - val_accuracy: 0.9849 Epoch 7/10 750/750 [==] - 61s 81ms/step - loss: 0.0454 - accuracy: 0.9848 - val_loss: 0.0499 - val_accuracy: 0.9868 Epoch 8/10 750/750 [=== - 60s 80ms/step - loss: 0.0383 - accuracy: 0.9876 - val_loss: 0.0494 - val_accuracy: 0.9870 Epoch 9/10 750/750 [==] - 61s 82ms/step - loss: 0.0369 - accuracy: 0.9875 - val_loss: 0.0507 - val_accuracy: 0.9860 Epoch 10/10 750/750 [==] - 61s 82ms/step - loss: 0.0329 - accuracy: 0.9885 - val_loss: 0.0473 - val_accuracy: 0.9868

Neural Network Prediction Results

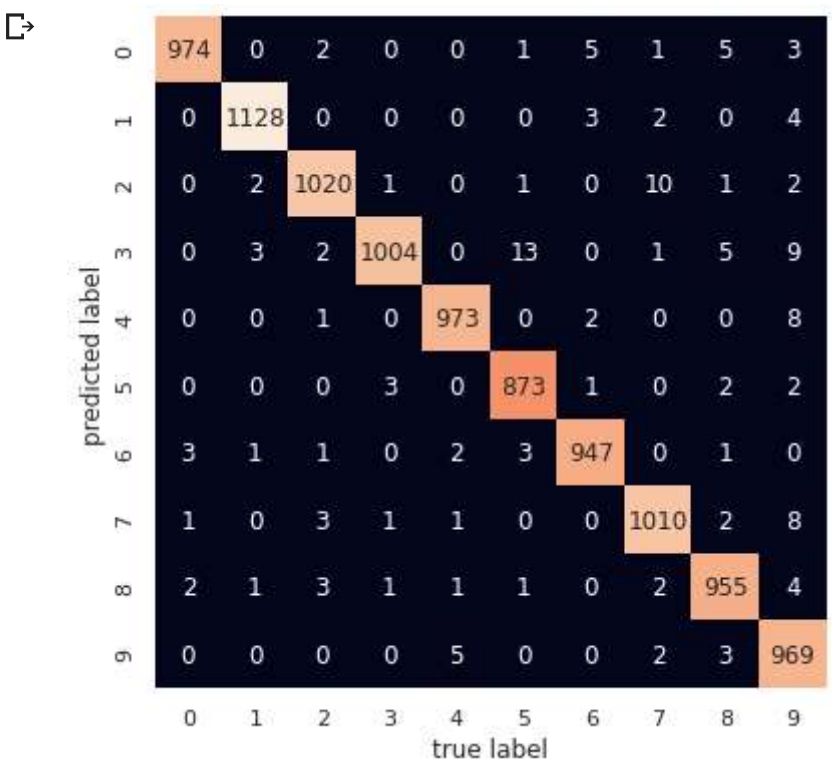
```
# Convert nn_model prediction results
nnt_prediction_results = nnt_model.predict(nnt_x_test)
nnt_y_fit = np.full((10000,), 0, dtype='uint8')

for i in range(0, len(nnt_prediction_results)-1, 1):
  nnt_y_fit[i] = np.argmax(nnt_prediction_results[i])
```

```
print(classification_report(y_test, nnt_y_fit, target_names=label_name))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.99      0.99      1032
           3       0.97      0.99      0.98      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.99      0.96      0.97      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

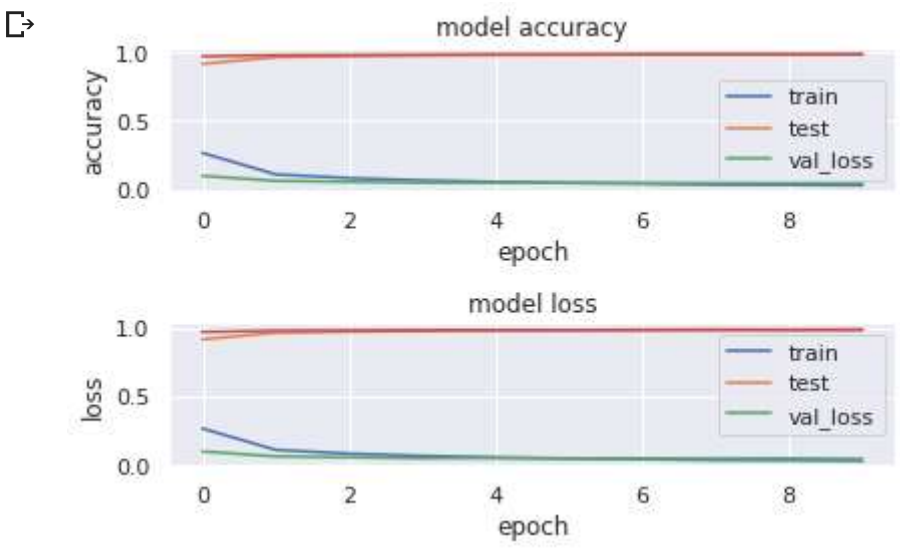
Neural Network Confusion Matrix

```
mat = confusion_matrix(y_test, nnt_y_fit)
plt.figure(figsize=(6, 10))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=label_name,
            yticklabels=label_name)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Plot Accuracy and loss

```
# plotting the metrics
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(hist.history['loss'])
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_loss'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test','val_loss'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(hist.history['loss'])
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_loss'])
plt.plot(hist.history['val_accuracy'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test','val_loss'], loc='upper right')

plt.tight_layout()
```



Neural Network Result

The number that has the highest miss-prediction true label nine predicted as four, but overall, Neural Network has the highest accuracy of 99%.

Overall analysis on the following model GaussianNB, KNeighbors(knn) and RandomForest.

GaussianNB rbf has overall accurary of 97% it is 3% more better than the model mentioned above. Run time 935 ms very resonalble about of time.

Comparing one algorithm over another is not entirely straightforward. Various considerations need to be accounted for, including validation accuracy and runtime performance. Also, the provided dataset is not always enough, and sometimes requiring the need for further partitioning.

Conclusion

There are many different approaches, and packaged tools to allow end users to plug a dataset and determine a prediction result. Some of these off the shelf solutions, require more effort than others. However, these tools ultimately streamline, and make analysis much easier. Therefore, it may become too easy to regularly depend on one technique, or a fixed set of packages. This can be dangerous, as results have shown that choosing one model over the other requires careful choice, and knowledge regarding the shape of the dataset(s). Comparing models should not be a binary comparison whether one generates a better result, or one has a better runtime. Instead users need to holistically determine an acceptable error tolerance, while understanding their computing requirements. For example, having unrealistic expectation of running a dataset through a more computationally expensive algorithm, will prove counterproductive. Therefore, users would need to consider options of subsampling, or acquiring different computing resources. Combining this knowledge can greatly increase prediction accuracy, while decreasing the prediction times.