

## **1. Data preparation, exploration, visualization**

In this assignment, not much needed to be done in terms of preparing the data as much of the data was image data. However, when creating the principal components, I did need to standardize the data as PCA is affected by scale and therefore, the features in the data needed to be scaled prior to applying PCA. To standardize the data, I used StandardScaler and performed a transformation on both the training and test sets. Min-Max scaling could have been used as well. When performing PCA, I created a plot in order to visualize the digit classes. When performing the train-test split, I created plots of validation curves to better understand how the training and cross validation scores differed based on the parameters that were used

## **2. Review research design and modeling methods**




This assignment entailed using methods such as Random Forest, Principal Component Analysis and Train Test Split. For the Random Forest, I used the XGBRFClassifier and the metrics that I used for evaluating the performance was the F1 Score. After running the Random Forest Classifier, I got a F1 Score of 0.94375 which was good. In addition, I generated the confusion matrix. I also performed PCA to generate the principal components that represent 95 percent of the variability in the explanatory variables. PCA is a linear space transformation where the first component captures the highest variation in the data, the second component the second highest variation and so on. I obtained 429 components that captured 95% of the variation. Once I got the principal components, I then constructed another random forest classifier and again calculated the F1 Score, which was 0.9517. The biggest design flaw in this experiment has to do with not creating a training and validation set within the training data. I then reran the experiment such that it was consistent with the train-test split regimen used in past assignments. The train-test split did yield a better F1 score of 0.969. Finally, I performed k-means clustering to group the MNIST observations into one of the ten observations and then assigned labels. Compared to the results from Random Forest and Train-Test split, the results from the k-means clustering were not great as the accuracy was only 64%. When submitting to Kaggle, I ran into issues when performing the train-test split. Specifically, Kaggle had errors presumably because I was using a GPU to speed up the computation. Because I had already run GridSearch, I knew the optimal parameters and therefore, skipped the GridSearch as part of submitting to Kaggle. classifier. I was also able to graph the validation curves to better understand how the training and cross validation scores differed based on the parameters that were used.

### 3. Review results, evaluate models

Based on the results alone, I do think that PCA, Random Forest Classifiers, and Train-Test split perform much better than clustering. However, compared to PCA and Random Forest, the Train-Test split performed better. When submitted to Kaggle, the train test split resulted in a score of 0.89780, principal components analysis resulted in a score of 0.85920, and random forests resulted in a score of 0.83760. Random Forests, PCA, and Train Test Split all seemed to perform relatively well and with a GPU, the train-test split seemed to perform even faster. Running these algorithms on my computer was considerably slower, which restricted the number of times I could repeat the experiment.

### 4. Kaggle Submission Relative to Peers

Username – anaswarj

<b>MSDS 422 - Kannada MNIST PCA RF V2</b> (version 2/2) 18 hours ago by <a href="#">Anaswar Jayakumar</a> From Notebook [Kannada MNIST PCA RF V2]	Succeeded 	0.85920	0.85940	<input type="checkbox"/>
<b>MSDS 422 - Kannada MNIST RF</b> (version 5/5) 20 hours ago by <a href="#">Anaswar Jayakumar</a> From Notebook [Kannada MNIST RF]	Succeeded 	0.83760	0.83760	<input type="checkbox"/>
<b>Kannada Train Test Split</b> (version 9/9) 5 hours ago by <a href="#">Anaswar Jayakumar</a> From Notebook [Kannada Train Test Split]	Succeeded 	0.89780	0.89520	<input type="checkbox"/>

### 5. Exposition, problem description and management recommendations

I would recommend using PCA as a preliminary to machine learning classification. Before PCA was applied to create a new random forest classifier, the creation of the first random forest classifier already yielded decent results. When PCA was applied to create a new random forest classifier, the results improved. Moreover, PCA is a dimensionality reduction algorithm and when working with thousands of features, a dimensionality reduction algorithm such as PCA does play an important role. Lastly, PCA can be used to improve the outcome of a classifier. For example, when PCA was applied to the original random forest classifier, the F1 score improved from 0.94375 to 0.9517. Even though this isn't a significant improvement, it is still an improvement nonetheless and given enough time, I am sure PCA would continue to improve the F1 score of the original random forest classifier

# MSDS 422 - Assignment 5 Final

May 2, 2021

## 1 Assignment 5: Principal Components Analysis & Clustering

You will compete in the Kaggle.com Kannada Digit Recognizer competition which involves classical digit recognition from hand-written images.

Read the competition rules, and download the MNIST training and test set data. This binary classification task is NOT what is required for the current assignment. In this assignment we are asking for a multiclass classifier. The entire MNIST data set will be used for input data. For this assignment, you will develop a classifier that may be used to predict which of the 10 digits is being written.

- (1) Begin by fitting a random forest classifier using the full set of 784 explanatory variables and the model training set (train.csv). Record the time it takes to fit the model and then evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (2) Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the 784 explanatory variables. Record the time it takes to identify the principal components.
- (3) Using the identified principal components from step (2), use the train.csv to build another random forest classifier. Record the time it takes to fit the model and to evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (4) Submit both the RF Classifier and the PCA RF Classifier to Kaggle.com, and report both scores along with your user name. I MUST have your user name to verify submission status.
- (5) The experiment we have proposed has a MAJOR design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com. Provide your Kaggle.com score and user ID.
- (6) Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels. (Follow the example here if needed: [kmeans mnist.pdf](#) download)

Report total elapsed time measures for the training set analysis. It is sufficient to run a single time-elapsed test for this assignment. In practice, we might consider the possibility of repeated executions of the relevant portions of the programs, much as the Benchmark Example programs do. Some code that might help you with reporting elapsed total time follows.

```
start=datetime.now()
rf2.fit(trainimages,labels)
end=datetime.now()
print(end-start)
```

Relevant scikit-learn documentation includes:

Random Forest Classifier Metrics Classification Report Plot Digits Classification Sklearn Decomposition

(Optional reading) If you want to learn about the time it takes to execute individual functions or code segments within Python, this article demonstrates a variety of ways to do it

Regarding the F1 score and the evaluation of multiclass classifiers, refer to the literature on information retrieval. See pages 142–145 of this classic reference:

Manning, C. D., Raghaven, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge, UK: Cambridge University Press. [ISBN-13: 978-0521865715]

Or see pages 154–158 of the free online version of the book here

Additional information about this book is available online here

Programming Notes

One of the key parameters in setting up random forests is the number of explanatory variables to include in the individual trees. For this classification problem, I would suggest that we follow the advice of Müller and Guido (2017) and use `max_features = 'sqrt'`.

Regarding the other meta-parameters ... ensure that `bootstrap = True` and, given the large number of observations, we might as well keep the default value of `n_estimators = 10`.

Müller, A. C., & Guido, S. (2017). Introduction to machine learning with Python: A guide for data scientists. Sebastopol, CA: O'Reilly. [ISBN-13: 978-1449369415]. Code examples here

Another useful reference that discusses the MNIST data set and principal components analysis is:

VanderPlas, J. (2017). Python data science handbook: Essential tools for working with data. Sebastopol, CA: O'Reilly [ISBN-13: 978-1491912058].

Management Problem

From a management perspective, the predictive accuracy of models must be weighed against the costs of model development and implementation. Suppose you were the manager of a data science team responsible for implementing models for computer vision (classification of images analogous to the MNIST problem). Would you recommend using PCA as a preliminary to machine learning classification? Explain your thinking.

Grading Guidelines (50 points)

- (1) Data preparation, exploration, visualization (10 points)
- (2) Review research design and modeling methods (10 points)
- (3) Review results, evaluate models (10 points)

(4) Implementation and programming (10 points)

(5) Exposition, problem description, and management recommendations (10 points)

#### Deliverables and File Formats

Provide a double-spaced paper with a two-page maximum for the text. The paper should include all a discussion of all graded elements but focus particularly on insights. Include your Python code and output as an appendix. Upload this as a single .pdf file.

#### Formatting Python Code

Refer to Google's Python Style Guide for ideas about formatting Python code

Comment often and in detail, highlighting major sections of code, describing the thinking behind the modeling and programming methods being employed.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

```
[503]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6

import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

```
[504]: df_train = pd.read_csv('/Users/anaswarjayakumar/Downloads/train 3.csv')
df_test = pd.read_csv('/Users/anaswarjayakumar/Downloads/test 2.csv')
```

```
[505]: df_train.head()
```

```
[505]:
```

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

	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	\
0	0	...	0	0	0	0	0	0	
1	0	...	0	0	0	0	0	0	
2	0	...	0	0	0	0	0	0	

3	0	...	0	0	0	0	0	0
4	0	...	0	0	0	0	0	0

	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

[5 rows x 785 columns]

```
[506]: df_train['label'].dtype
```

```
[506]: dtype('int64')
```

```
[507]: type(df_train['label'])
```

```
[507]: pandas.core.series.Series
```

```
[508]: df_train['label'].unique()
```

```
[508]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[509]: df_test.head()
```

```
[509]:
```

	id	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	\
0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	0	
3	3	0	0	0	0	0	0	0	0	0	
4	4	0	0	0	0	0	0	0	0	0	

	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	\
0	...	0	0	0	0	0	0	0	
1	...	0	0	0	0	0	0	0	
2	...	0	0	0	0	0	0	0	
3	...	0	0	0	0	0	0	0	
4	...	0	0	0	0	0	0	0	

	pixel781	pixel782	pixel783
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

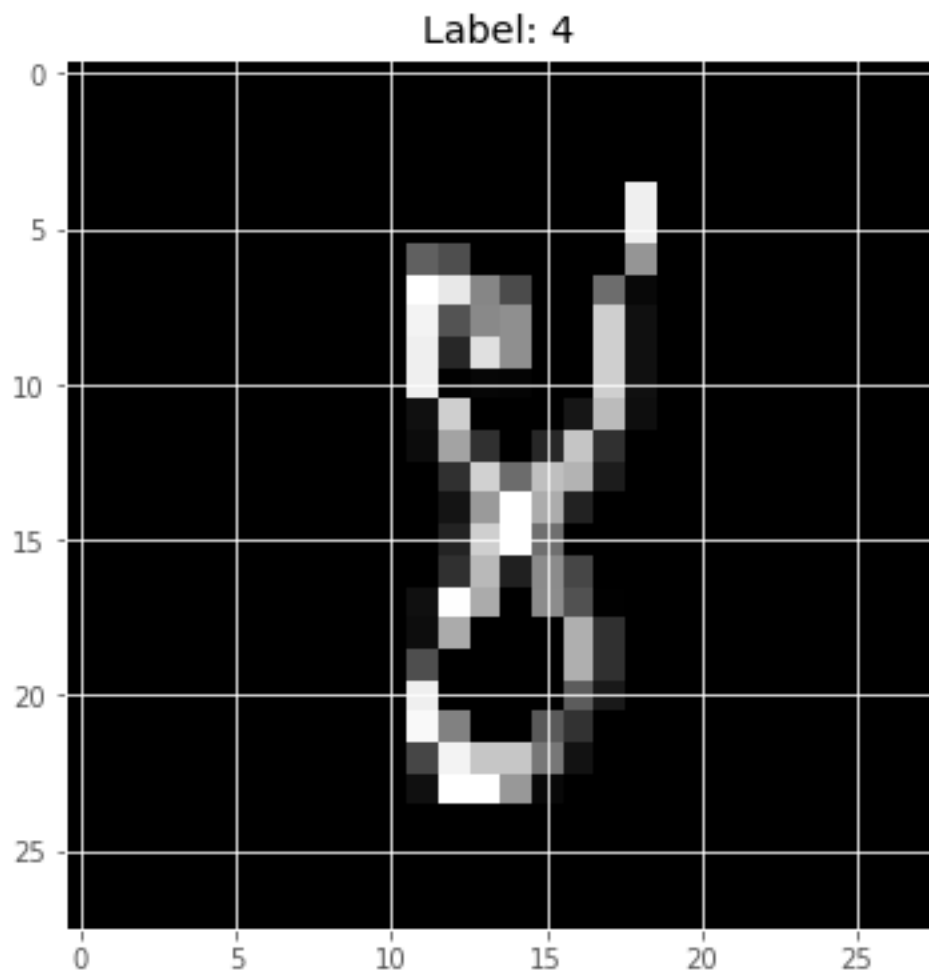
[5 rows x 785 columns]

```
[510]: print(df_train.shape)
print(df_test.shape)
```

(60000, 785)

(5000, 785)

```
[511]: num = 4
plot_num = df_train.iloc[num, 1:]
plot_num = np.array(plot_num).reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.title(f'Label: {df_train.iloc[num, 0]}')
plt.show()
```



## 2 Step 1 - Random Forest Classifier

```
[344]: df_train.values
```

```
[344]: array([[0, 0, 0, ..., 0, 0, 0],
          [1, 0, 0, ..., 0, 0, 0],
          [2, 0, 0, ..., 0, 0, 0],
          ...,
          [7, 0, 0, ..., 0, 0, 0],
          [8, 0, 0, ..., 0, 0, 0],
          [9, 0, 0, ..., 0, 0, 0]])
```

```
[345]: df_test.values
```

```
[345]: array([[ 0,  0,  0, ...,  0,  0,  0],
          [ 1,  0,  0, ...,  0,  0,  0],
          [ 2,  0,  0, ...,  0,  0,  0],
          ...,
          [4997,  0,  0, ...,  0,  0,  0],
          [4998,  0,  0, ...,  0,  0,  0],
          [4999,  0,  0, ...,  0,  0,  0]])
```

```
[346]: df_train.values.shape
```

```
[346]: (60000, 785)
```

```
[347]: df_test.values.shape
```

```
[347]: (5000, 785)
```

```
[512]: X = df_train.iloc[:, 1:].values
      X
```

```
[512]: 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]])
```

```
[513]: X.shape
```

```
[513]: (60000, 784)
```

```
[514]: y = df_train.iloc[:, 0].values
      y
```



```
[514]: array([0, 1, 2, ..., 7, 8, 9])
```

```
[515]: y.shape
```

```
[515]: (60000,)
```

```
[452]: import xgboost as xgb
from xgboost import XGBRFClassifier

param_grid = {
    'eta': [0.05],
    'n_estimators': [500, 1000],
    'colsample_bytree': [0.7, 0.8],
    'max_depth': [6, 10],
    'reg_alpha': [0, 1],
    'reg_lambda': [1, 2],
    'subsample': [0.5, 0.9]
}
```

```
[453]: xgb_rf = XGBRFClassifier(n_estimators=50, learning_rate=0.05, subsample=0.7,
    ↪ colsample_bytree=0.7,
    max_depth=6, reg_alpha=0, eval_metric='mlogloss')

%time xgb_rf.fit(X, y)
```

CPU times: user 9min 29s, sys: 2.2 s, total: 9min 31s

Wall time: 9min 32s

```
[453]: XGBRFClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
    colsample_bytree=0.7, eval_metric='mlogloss', gamma=0,
    gpu_id=-1, importance_type='gain', interaction_constraints='',
    learning_rate=0.05, max_delta_step=0, max_depth=6,
    min_child_weight=1, missing=nan, monotone_constraints='()',
    n_estimators=50, n_jobs=1, num_parallel_tree=50,
    objective='multi:softprob', random_state=0, reg_alpha=0,
    scale_pos_weight=None, subsample=0.7, tree_method='exact',
    validate_parameters=1, verbosity=None)
```

```
[454]: %time y_pred = xgb_rf.predict(X)
y_pred
```

CPU times: user 1.03 s, sys: 203 ms, total: 1.23 s

Wall time: 1.24 s

```
[454]: array([0, 1, 2, ..., 7, 8, 9])
```

```
[455]: from sklearn.metrics import confusion_matrix

confusion_matrix(y, y_pred)
```

```
[455]: array([[5703, 185, 0, 43, 4, 6, 9, 23, 17, 10],
[ 30, 5859, 1, 33, 8, 11, 2, 9, 13, 34],
[ 53, 15, 5840, 42, 1, 10, 14, 22, 3, 0],
[ 57, 26, 0, 5621, 125, 64, 15, 89, 3, 0],
[ 11, 9, 1, 108, 5736, 93, 1, 15, 6, 20],
[ 0, 18, 3, 47, 130, 5758, 3, 20, 10, 11],
[ 8, 33, 0, 72, 9, 7, 5614, 246, 11, 0],
[ 62, 36, 0, 202, 180, 15, 228, 5269, 0, 8],
[ 81, 17, 0, 17, 53, 69, 4, 19, 5644, 96],
[ 42, 22, 0, 14, 186, 8, 42, 69, 101, 5516]])
```

```
[456]: from sklearn.metrics import f1_score
print(f1_score(y, y_pred, average='micro'))
```

0.9426666666666667

```
[457]: import pandas as pd
import numpy as np
from scipy import interp

from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import LabelBinarizer

def class_report(y_true, y_pred, y_score=None, average='micro'):
    if y_true.shape != y_pred.shape:
        print("Error! y_true %s is not the same shape as y_pred %s" % (
            y_true.shape,
            y_pred.shape)
        )
        return

    lb = LabelBinarizer()

    if len(y_true.shape) == 1:
        lb.fit(y_true)

    #Value counts of predictions
    labels, cnt = np.unique(
        y_pred,
        return_counts=True)
    n_classes = len(labels)
    pred_cnt = pd.Series(cnt, index=labels)

    metrics_summary = precision_recall_fscore_support(
        y_true=y_true,
        y_pred=y_pred,
        labels=labels)
```

```

avg = list(precision_recall_fscore_support(
    y_true=y_true,
    y_pred=y_pred,
    average='weighted'))

metrics_sum_index = ['precision', 'recall', 'f1-score', 'support']
class_report_df = pd.DataFrame(
    list(metrics_summary),
    index=metrics_sum_index,
    columns=labels)

support = class_report_df.loc['support']
total = support.sum()
class_report_df['avg / total'] = avg[:-1] + [total]

class_report_df = class_report_df.T
class_report_df['pred'] = pred_cnt
class_report_df['pred'].iloc[-1] = total

if not (y_score is None):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for label_it, label in enumerate(labels):
        fpr[label], tpr[label], _ = roc_curve(
            (y_true == label).astype(int),
            y_score[:, label_it])

        roc_auc[label] = auc(fpr[label], tpr[label])

    if average == 'micro':
        if n_classes <= 2:
            fpr["avg / total"], tpr["avg / total"], _ = roc_curve(
                lb.transform(y_true).ravel(),
                y_score[:, 1].ravel())
        else:
            fpr["avg / total"], tpr["avg / total"], _ = roc_curve(
                lb.transform(y_true).ravel(),
                y_score.ravel())

        roc_auc["avg / total"] = auc(
            fpr["avg / total"],
            tpr["avg / total"])

    elif average == 'macro':
        # First aggregate all false positive rates

```

```

all_fpr = np.unique(np.concatenate([
    fpr[i] for i in labels]
))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in labels:
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr

roc_auc["avg / total"] = auc(fpr["macro"], tpr["macro"])

class_report_df['AUC'] = pd.Series(roc_auc)

return class_report_df

```

```

[458]: report_with_auc = class_report(
    y_true=y.ravel(),
    y_pred=xgb_rf.predict(X),
    y_score=xgb_rf.predict_proba(X))

print(report_with_auc)

```

	precision	recall	f1-score	support	pred	AUC
0	0.943112	0.950500	0.946792	6000.0	6047.0	0.995406
1	0.941961	0.976500	0.958920	6000.0	6220.0	0.997767
2	0.999145	0.973333	0.986070	6000.0	5845.0	0.998013
3	0.906759	0.936833	0.921551	6000.0	6199.0	0.994683
4	0.891791	0.956000	0.922780	6000.0	6432.0	0.997156
5	0.953153	0.959667	0.956399	6000.0	6041.0	0.997724
6	0.946392	0.935667	0.940999	6000.0	5932.0	0.995524
7	0.911434	0.878167	0.894491	6000.0	5781.0	0.988301
8	0.971763	0.940667	0.955962	6000.0	5808.0	0.996308
9	0.968569	0.919333	0.943309	6000.0	5695.0	0.993725
avg / total	0.943408	0.942667	0.942727	60000.0	60000.0	0.995568

### 3 Step 1 Submission

```

[516]: X_test = df_test.iloc[:, 1:].values
X_test

```

```
[516]: 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]])
```

```
[517]: y_pred_test = xgb_rf.predict(X_test)
```

```
[492]: submission_data = np.c_[df_test["id"].values, y_pred_test]
submission_df = pd.DataFrame(data = submission_data, columns = ["id", "label"])
submission_df['id'] = submission_df['id'].astype('int64')
submission_df['label'] = submission_df['label'].astype('int64')
submission_df
```

```
[492]:
```

	id	label
0	0	3
1	1	0
2	2	2
3	3	6
4	4	7
...	...	...
4995	4995	1
4996	4996	1
4997	4997	1
4998	4998	6
4999	4999	3

[5000 rows x 2 columns]

```
[478]: submission_df.to_csv("/Users/anaswarjayakumar/Downloads/kannada_mnist_rf.csv",
    ↪index = False)
```

## 4 Step 2 - Principal Component Analysis for the combined training and test

### 5 data

#### 5.1 Standardize the Data

```
[518]: from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X)
```

```
# Apply transform to both the training set and the test set.
train_img = scaler.transform(X)
test_img = scaler.transform(X_test)
```

## 5.2 Import and Apply PCA

```
[519]: from sklearn.decomposition import PCA
```

```
# Make an instance of the Model
pca = PCA(.95)
```

```
[520]: pca.fit(train_img)
```

```
[520]: PCA(n_components=0.95)
```

```
[521]: pca_components = pca.n_components_
pca_components
```

```
[521]: 429
```

```
[522]: import plotly.express as px
from sklearn.decomposition import PCA

pca_scatter = PCA(n_components=2)
components = pca_scatter.fit_transform(df_train)

principalDf = pd.DataFrame(data = components, columns = ['PC 1', 'PC 2'])
finalDf = pd.concat([principalDf, df_train[['label']]], axis = 1)
```

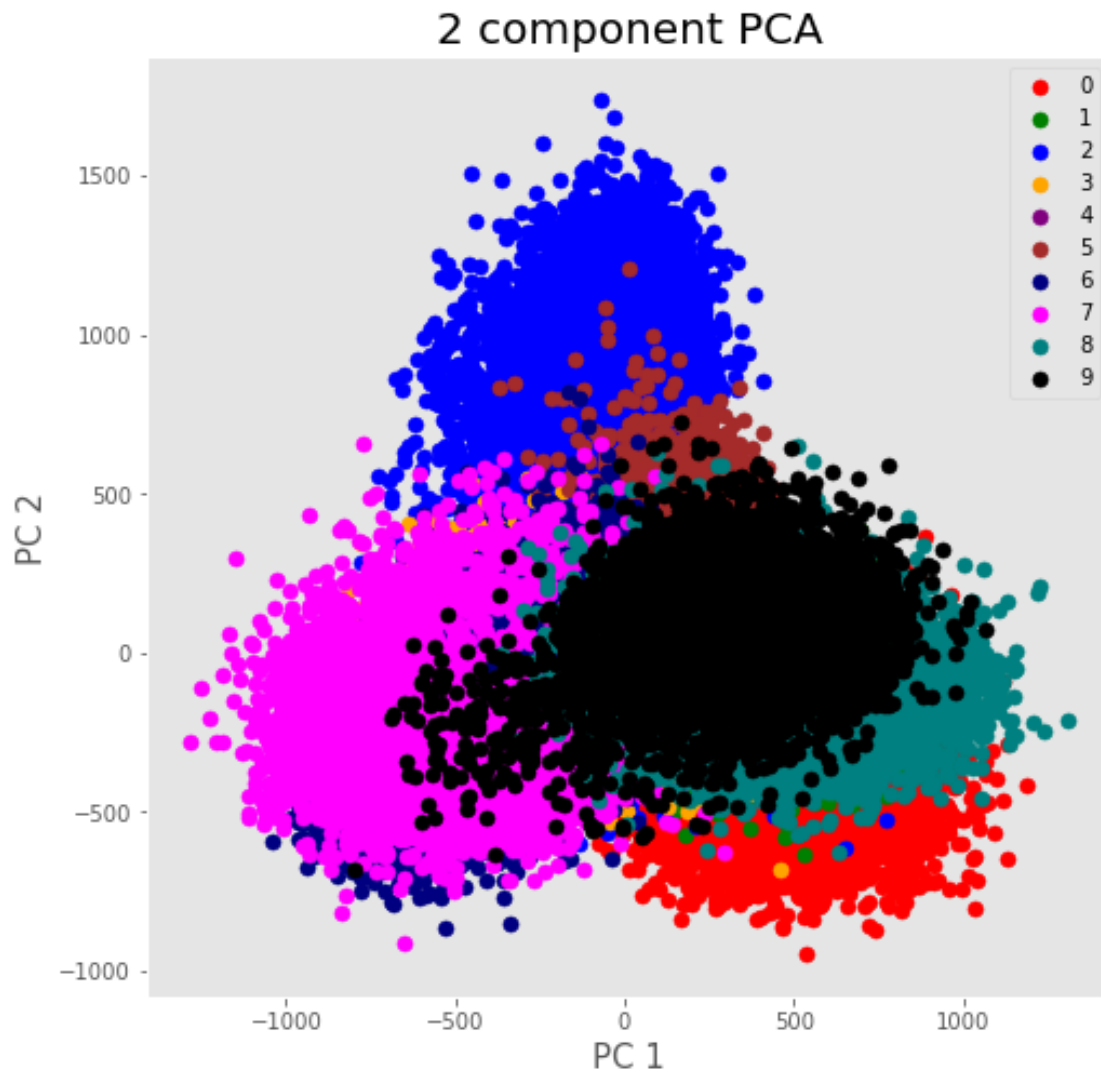
```
[523]: fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)

ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
colors = ['red', 'green', 'blue', 'orange', 'purple', 'brown', 'navy', 'magenta', 'teal', 'black']

for target, color in zip(targets, colors):
    indicesToKeep = finalDf['label'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'PC 1'],
               finalDf.loc[indicesToKeep, 'PC 2'],
               c = color,
               s = 50)
```

```
ax.legend(targets)
ax.grid()
```



6 Step 3 - Using the identified principal components from step 2, use the

7 train.csv to build another random forest classifier.

```
[524]: train_img = pca.transform(train_img)
```

```
[525]: train_img.shape
```

[525]: (60000, 429)

```
[526]: test_img = pca.transform(test_img)
```

```
[527]: test_img.shape
```

[527]: (5000, 429)

```
[223]: import xgboost as xgb
from xgboost import XGBRFClassifier

xgb_rf_model = XGBRFClassifier()

param_grid = {
    'eta': [0.05],
    'n_estimators': [500, 1000],
    'colsample_bytree': [0.7, 0.8],
    'max_depth': [6, 10],
    'reg_alpha': [0, 1],
    'reg_lambda': [1, 2],
    'subsample': [0.5, 0.9]
}
```

```
[225]: xgb_rf_train_img = XGBRFClassifier(n_estimators=50, learning_rate=0.05,
    ↪subsample=0.7, colsample_bytree=0.7,
        max_depth=6, reg_alpha=0, eval_metric='mlogloss')

%time xgb_rf_train_img.fit(train_img, y)
```

CPU times: user 7min 18s, sys: 1.34 s, total: 7min 19s

Wall time: 7min 22s

```
[225]: XGBRFClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
    colsample_bytree=0.7, eval_metric='mlogloss', gamma=0,
    gpu_id=-1, importance_type='gain', interaction_constraints='',
    learning_rate=0.05, max_delta_step=0, max_depth=6,
    min_child_weight=1, missing=nan, monotone_constraints='()',
    n_estimators=50, n_jobs=1, num_parallel_tree=50,
    objective='multi:softprob', random_state=0, reg_alpha=0,
    scale_pos_weight=None, subsample=0.7, tree_method='exact',
    validate_parameters=1, verbosity=None)
```

```
[226]: from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

%time y_pred_train_img = xgb_rf_train_img.predict(train_img)
y_pred_train_img
```



CPU times: user 622 ms, sys: 35 ms, total: 656 ms  
Wall time: 660 ms

```
[226]: array([0, 1, 2, ..., 7, 8, 9])
```

```
[227]: from sklearn.metrics import confusion_matrix

confusion_matrix(y, y_pred_train_img)
```

```
[227]: array([[5488, 322, 8, 46, 6, 3, 3, 8, 95, 21],
 [ 81, 5807, 2, 49, 10, 6, 1, 1, 17, 26],
 [ 48, 7, 5911, 14, 2, 7, 1, 5, 3, 2],
 [ 75, 4, 14, 5724, 56, 50, 18, 52, 4, 3],
 [ 1, 2, 1, 48, 5779, 132, 9, 1, 10, 17],
 [ 0, 5, 7, 71, 95, 5802, 4, 5, 11, 0],
 [ 3, 4, 7, 60, 22, 11, 5692, 185, 0, 16],
 [ 24, 10, 6, 126, 51, 5, 375, 5374, 4, 25],
 [142, 12, 6, 35, 49, 21, 2, 3, 5682, 48],
 [ 16, 3, 3, 32, 98, 10, 70, 62, 38, 5668]])
```

```
[228]: from sklearn.metrics import f1_score
print(f1_score(y, y_pred_train_img, average='micro'))
```

0.9487833333333333

```
[229]: report_with_auc = class_report(
    y_true=y.ravel(),
    y_pred=xgb_rf_train_img.predict(train_img),
    y_score=xgb_rf_train_img.predict_proba(train_img))

print(report_with_auc)
```

	precision	recall	f1-score	support	pred	AUC
0	0.933651	0.914667	0.924061	6000.0	5878.0	0.994849
1	0.940253	0.967833	0.953844	6000.0	6176.0	0.998023
2	0.990947	0.985167	0.988048	6000.0	5965.0	0.999158
3	0.922482	0.954000	0.937976	6000.0	6205.0	0.997452
4	0.936933	0.963167	0.949869	6000.0	6168.0	0.998775
5	0.959484	0.967000	0.963227	6000.0	6047.0	0.998644
6	0.921781	0.948667	0.935031	6000.0	6175.0	0.996663
7	0.943469	0.895667	0.918947	6000.0	5696.0	0.996068
8	0.968963	0.947000	0.957856	6000.0	5864.0	0.997965
9	0.972880	0.944667	0.958566	6000.0	5826.0	0.997115
avg / total	0.949084	0.948783	0.948742	60000.0	60000.0	0.997572

## 8 Step 5 - Rerun the experiment in a way that is consistent with a training-

## 9 and-test regimen

```
[386]: from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=1/7.0,
↪random_state=0)
print(X_train.shape)
print(X_val.shape)
```

(51428, 784)

(8572, 784)

```
[459]: def algorithm_pipeline(X_train_data, X_test_data, y_train_data, y_test_data,
                             model, param_grid, cv=10, scoring_fit='f1_micro'):
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit,
        verbose=2,
        return_train_score=True # set this for train score
    )

    fitted_model = gs.fit(X_train_data, y_train_data.ravel())
    pred = fitted_model.predict(X_test_data)

    return gs, fitted_model, pred
```

```
[470]: import xgboost as xgb

xgb_class_model = xgb.XGBRFClassifier()

param_grid = {
    'n_estimators': [10, 25, 50],
    'colsample_bytree': [0.6],
    'max_depth': [6, 10, 15],
    'subsample': [0.6]
}

%time gs, xgb_class_model, pred = algorithm_pipeline(X_train, X_val, y_train,
↪y_val, xgb_class_model, \
                                                    param_grid, cv=3)
```

Fitting 3 folds for each of 9 candidates, totalling 27 fits

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

[Parallel(n\_jobs=-1)]: Done 27 out of 27 | elapsed: 29.2min finished

```
[18:31:48] WARNING: /Users/anaswarjayakumar/xgboost/python-
package/build/temp.macosx-10.9-x86_64-3.8/xgboost/src/learner.cc:1094: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective
'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
CPU times: user 17min 26s, sys: 4.2 s, total: 17min 30s
Wall time: 46min 41s
```

```
[471]: print(xgb_class_model.best_params_)
```

```
{'colsample_bytree': 0.6, 'max_depth': 15, 'n_estimators': 50, 'subsample': 0.6}
```

```
[472]: df = pd.DataFrame(gs.cv_results_)
df
```

```
[472]:
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
0	90.493725	0.705474	0.309051	0.042505	
1	219.928849	0.570571	0.396026	0.048062	
2	439.357982	1.099870	0.456482	0.063880	
3	145.987790	1.167988	0.377782	0.030502	
4	367.619038	0.397554	0.564153	0.019914	
5	737.039762	5.848956	0.793618	0.154009	
6	213.394001	0.560373	0.515698	0.016342	
7	523.952525	6.718039	0.614626	0.058860	
8	814.169167	36.939717	0.976694	0.159563	

	param_colsample_bytree	param_max_depth	param_n_estimators	param_subsample	\
0	0.6	6	10	0.6	
1	0.6	6	25	0.6	
2	0.6	6	50	0.6	
3	0.6	10	10	0.6	
4	0.6	10	25	0.6	
5	0.6	10	50	0.6	
6	0.6	15	10	0.6	
7	0.6	15	25	0.6	
8	0.6	15	50	0.6	

	params	split0_test_score	\
0	{'colsample_bytree': 0.6, 'max_depth': 6, 'n_e...	0.927026	
1	{'colsample_bytree': 0.6, 'max_depth': 6, 'n_e...	0.931984	
2	{'colsample_bytree': 0.6, 'max_depth': 6, 'n_e...	0.932742	
3	{'colsample_bytree': 0.6, 'max_depth': 10, 'n_...	0.952342	
4	{'colsample_bytree': 0.6, 'max_depth': 10, 'n_...	0.957709	
5	{'colsample_bytree': 0.6, 'max_depth': 10, 'n_...	0.958467	
6	{'colsample_bytree': 0.6, 'max_depth': 15, 'n_...	0.959225	
7	{'colsample_bytree': 0.6, 'max_depth': 15, 'n_...	0.962900	
8	{'colsample_bytree': 0.6, 'max_depth': 15, 'n_...	0.965409	

	split1_test_score	split2_test_score	mean_test_score	std_test_score \
0	0.929826	0.928713	0.928521	0.001151
1	0.937876	0.932738	0.934199	0.002618
2	0.937351	0.935363	0.935152	0.001887
3	0.952925	0.952806	0.952691	0.000252
4	0.959109	0.957240	0.958019	0.000794
5	0.959984	0.958523	0.958991	0.000702
6	0.958934	0.960098	0.959419	0.000495
7	0.964359	0.963423	0.963561	0.000603
8	0.966342	0.964240	0.965330	0.000860

	rank_test_score	split0_train_score	split1_train_score \
0	9	0.941695	0.940265
1	8	0.946624	0.947703
2	7	0.946536	0.947849
3	6	0.972904	0.973574
4	5	0.976695	0.976987
5	4	0.977395	0.977045
6	3	0.983608	0.983200
7	2	0.985241	0.986233
8	1	0.986204	0.986233

	split2_train_score	mean_train_score	std_train_score
0	0.943505	0.941822	0.001325
1	0.946771	0.947033	0.000478
2	0.947296	0.947227	0.000538
3	0.973400	0.973293	0.000284
4	0.976900	0.976861	0.000122
5	0.977746	0.977396	0.000286
6	0.983842	0.983550	0.000265
7	0.985825	0.985767	0.000407
8	0.986175	0.986204	0.000024

```
[473]: results = ['mean_test_score',
                  'mean_train_score',
                  'std_test_score',
                  'std_train_score']

def pooled_var(stds):
    # https://en.wikipedia.org/wiki/Pooled\_variance#Pooled\_standard\_deviation
    n = 5 # size of each group
    return np.sqrt(sum((n-1)*(stds**2))/ len(stds)*(n-1))

fig, axes = plt.subplots(1, len(param_grid),
                        figsize = (5*len(param_grid), 7),
                        sharey='row')
axes[0].set_ylabel("Score", fontsize=25)
```

```

for idx, (param_name, param_range) in enumerate(param_grid.items()):
    grouped_df = df.groupby(f'param_{param_name}')[results]\
        .agg({'mean_train_score': 'mean',
              'mean_test_score': 'mean',
              'std_train_score': pooled_var,
              'std_test_score': pooled_var})

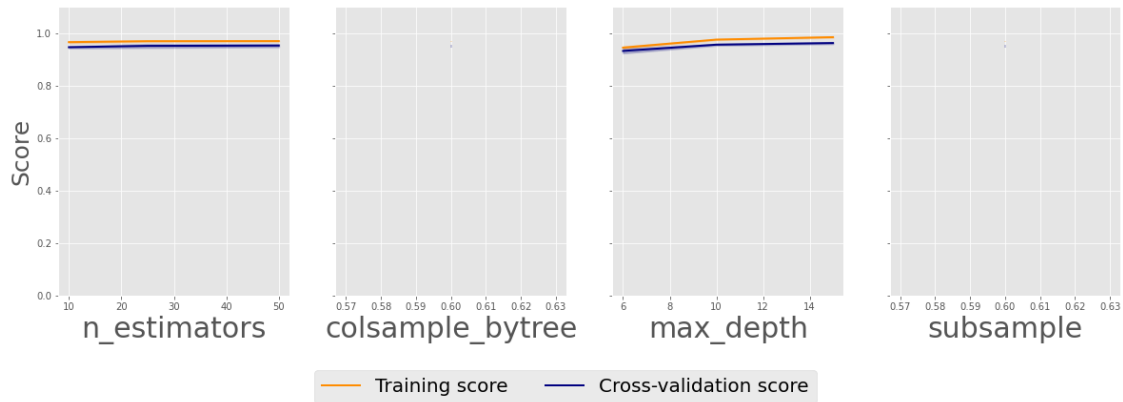
    previous_group = df.groupby(f'param_{param_name}')[results]
    axes[idx].set_xlabel(param_name, fontsize=30)
    axes[idx].set_ylim(0.0, 1.1)
    lw = 2
    axes[idx].plot(param_range, grouped_df['mean_train_score'], label="Training_
    ↪score",
                   color="darkorange", lw=lw)
    axes[idx].fill_between(param_range, grouped_df['mean_train_score'] -
    ↪grouped_df['std_train_score'],
                           grouped_df['mean_train_score'] +
    ↪grouped_df['std_train_score'], alpha=0.2,
                           color="darkorange", lw=lw)
    axes[idx].plot(param_range, grouped_df['mean_test_score'],
    ↪label="Cross-validation score",
                   color="navy", lw=lw)
    axes[idx].fill_between(param_range, grouped_df['mean_test_score'] -
    ↪grouped_df['std_test_score'],
                           grouped_df['mean_test_score'] +
    ↪grouped_df['std_test_score'], alpha=0.2,
                           color="navy", lw=lw)

handles, labels = axes[0].get_legend_handles_labels()
fig.suptitle('Validation curves', fontsize=40)
fig.legend(handles, labels, loc=8, ncol=2, fontsize=20)

fig.subplots_adjust(bottom=0.25, top=0.85)
plt.show()

```

## Validation curves



```
[474]: xgb_rf_class = XGBRFClassifier(colsample_bytree=0.6, max_depth=15,
    ↪n_estimators=50, subsample=0.6)
```

```
%time xgb_rf_class.fit(X_train, y_train.ravel())
```

[18:51:44] WARNING: /Users/anaswarjayakumar/xgboost/python-package/build/temp.macosx-10.9-x86\_64-3.8/xgboost/src/learner.cc:1094: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval\_metric if you'd like to restore the old behavior.  
CPU times: user 17min 22s, sys: 2.64 s, total: 17min 25s  
Wall time: 17min 26s

```
[474]: XGBRFClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
    colsample_bytree=0.6, gamma=0, gpu_id=-1,
    importance_type='gain', interaction_constraints='',
    max_delta_step=0, max_depth=15, min_child_weight=1, missing=nan,
    monotone_constraints='()', n_estimators=50, n_jobs=1,
    num_parallel_tree=50, objective='multi:softprob',
    random_state=0, reg_alpha=0, scale_pos_weight=None,
    subsample=0.6, tree_method='exact', validate_parameters=1,
    verbosity=None)
```

```
[410]: %time y_pred_val = xgb_rf_class.predict(X_val)
    y_pred_val
```

CPU times: user 129 ms, sys: 4.23 ms, total: 134 ms  
Wall time: 133 ms

```
[410]: array([8, 7, 3, ..., 3, 5, 0])
```

```
[425]: %time y_pred_train = xgb_rf_class.predict(X_train)
y_pred_train
```

CPU times: user 603 ms, sys: 4.79 ms, total: 608 ms  
Wall time: 607 ms

```
[425]: array([9, 9, 2, ..., 3, 7, 2])
```

```
[409]: confusion_matrix(y_val, y_pred_val)
```

```
[409]: array([[874, 23, 0, 4, 1, 0, 2, 1, 2, 3],
 [ 1, 872, 0, 1, 0, 1, 2, 0, 0, 1],
 [ 4, 0, 861, 4, 0, 0, 0, 0, 0, 0],
 [ 6, 1, 0, 765, 10, 9, 1, 9, 1, 0],
 [ 0, 0, 0, 7, 835, 3, 2, 2, 2, 4],
 [ 0, 0, 0, 6, 24, 822, 0, 2, 3, 0],
 [ 0, 2, 0, 4, 0, 1, 872, 15, 3, 1],
 [ 3, 0, 0, 30, 19, 3, 35, 768, 0, 2],
 [10, 0, 0, 1, 6, 8, 1, 1, 796, 5],
 [ 4, 0, 0, 2, 8, 1, 6, 6, 12, 776]])
```

```
[426]: confusion_matrix(y_train, y_pred_train)
```

```
[426]: array([[4944, 106, 0, 23, 1, 1, 7, 0, 3, 5],
 [ 11, 5087, 0, 4, 1, 3, 2, 0, 2, 12],
 [ 23, 3, 5080, 12, 3, 2, 2, 4, 2, 0],
 [ 27, 3, 0, 5111, 24, 13, 3, 16, 1, 0],
 [ 2, 1, 1, 14, 5105, 19, 1, 0, 0, 2],
 [ 0, 4, 1, 14, 18, 5098, 1, 5, 1, 1],
 [ 3, 6, 0, 47, 9, 4, 4986, 40, 4, 3],
 [17, 10, 0, 60, 48, 3, 54, 4939, 1, 8],
 [44, 13, 0, 10, 33, 12, 5, 5, 5025, 25],
 [ 7, 7, 0, 7, 50, 3, 56, 46, 40, 4969]])
```

```
[430]: print(f1_score(y_val, y_pred_val, average='micro'))
```

0.9613859076061596

**10 Step 6 - Use k-means clustering to group MNIST observations into 1 of 10**

**11 categories and then assign labels.**

## 11.1 Preprocessing the MNIST images

```
[481]: # convert each image to 1 dimensional array
kmeansX = np.copy(X)
kmeansY = np.copy(y)

# normalize the data to 0 - 1
kmeansX = kmeansX.astype(float) / 255.

print(kmeansX.shape)
print(kmeansX[0].shape)
```

(60000, 784)

(784,)

## 11.2 K-Means Clustering

```
[482]: from sklearn.cluster import MiniBatchKMeans

n_digits = len(np.unique(kmeansY))
print(n_digits)

# Initialize KMeans model
kmeans = MiniBatchKMeans(n_clusters = n_digits)

# Fit the model to the training data
kmeans.fit(kmeansX)
kmeans.labels_.shape
```

10

[482]: (60000,)

## 11.3 Assigning Cluster Labels

```
[483]: def infer_cluster_labels(kmeans, actual_labels):
    inferred_labels = {}

    for i in range(kmeans.n_clusters):
        # find index of points in cluster
        labels = []
        index = np.where(kmeans.labels_ == i)

        # append actual labels for each point in cluster
        labels.append(actual_labels[index])

        # determine most common label
        if len(labels[0]) == 1:
```



```

        counts = np.bincount(labels[0])
    else:
        counts = np.bincount(np.squeeze(labels))

    # assign the cluster to a value in the inferred_labels dictionary
    if np.argmax(counts) in inferred_labels:
        # append the new number to the existing array at this slot
        inferred_labels[np.argmax(counts)].append(i)
    else:
        # create a new array in this slot
        inferred_labels[np.argmax(counts)] = [i]

    # print(labels)
    #print('Cluster: {}, label: {}'.format(i, np.argmax(counts)))
return inferred_labels

def infer_data_labels(X_labels, cluster_labels):
    # empty array of len(X)
    predicted_labels = np.zeros(len(X_labels)).astype(np.uint8)

    for i, cluster in enumerate(X_labels):
        for key, value in cluster_labels.items():
            if cluster in value:
                predicted_labels[i] = key

    return predicted_labels

```

```

[484]: # test the infer_cluster_labels() and infer_data_labels() functions
cluster_labels = infer_cluster_labels(kmeans, kmeansY)
X_clusters = kmeans.predict(kmeansX)
predicted_labels = infer_data_labels(X_clusters, cluster_labels)

print(predicted_labels[:20])
print(kmeansY[:20])

```

```

[1 1 2 4 4 5 7 7 8 9 0 1 1 4 4 5 7 7 8 9]
[0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9]

```

## 11.4 Optimizing and Evaluating the Clustering Algorithm

### 11.5 Visualizing Cluster Centroids

```

[485]: # Initialize and fit KMeans algorithm
kmeans = MiniBatchKMeans(n_clusters = 36)
kmeans.fit(kmeansX)

# record centroid values
centroids = kmeans.cluster_centers_

```

```

# print(centroids[0].reshape(28, 28))

# reshape centroids into images
images = centroids.reshape(36, 28, 28)
images *= 255
images = images.astype(np.uint8)

# determine cluster labels
cluster_labels = infer_cluster_labels(kmeans, kmeansY)

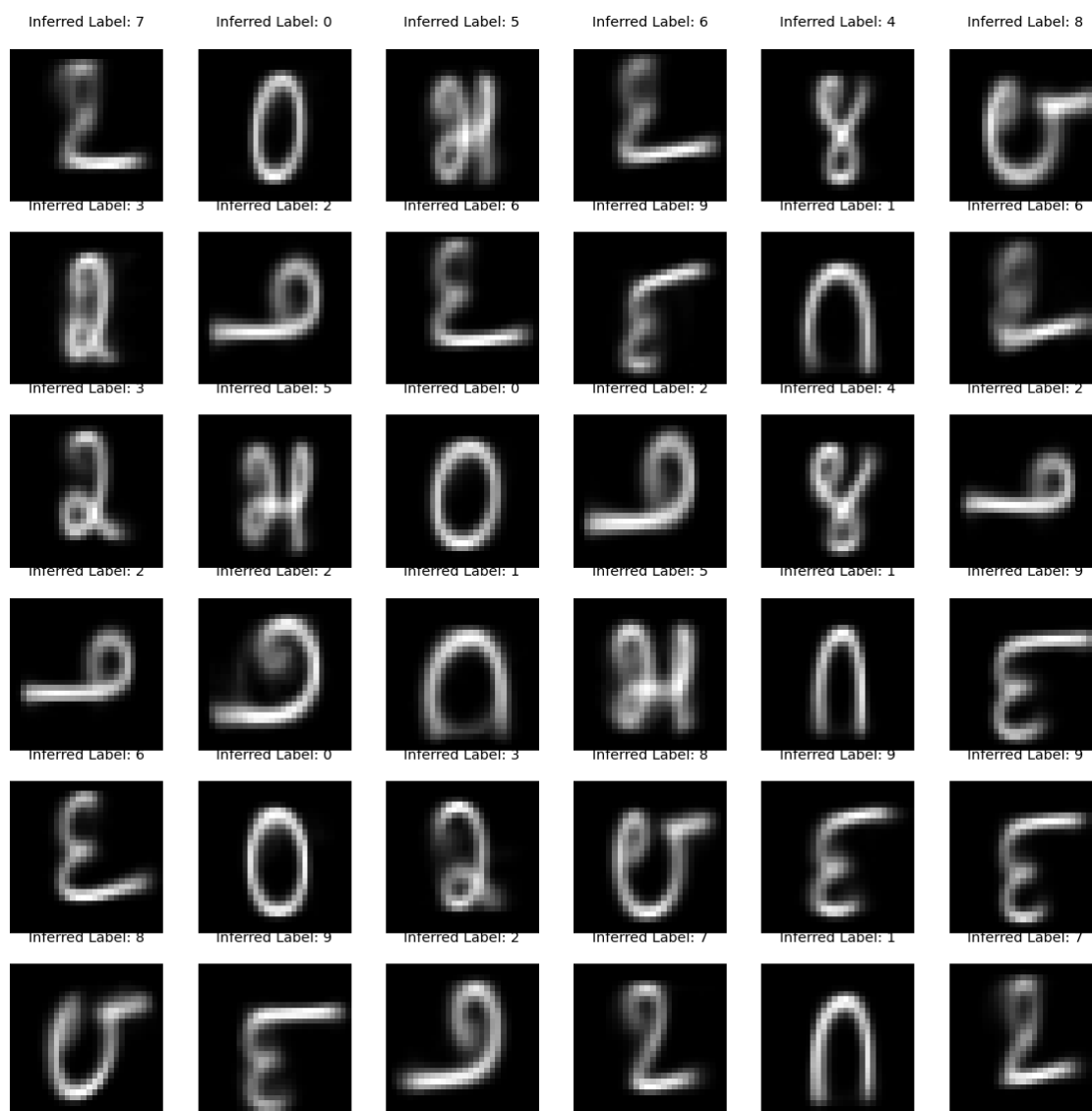
# create figure with subplots using matplotlib.pyplot
fig, axs = plt.subplots(6, 6, figsize = (20, 20))
plt.gray()

# loop through subplots and add centroid images
for i, ax in enumerate(axs.flat):
    # determine inferred label using cluster_labels dictionary
    for key, value in cluster_labels.items():
        if i in value:
            ax.set_title('Inferred Label: {}'.format(key))

    # add image to subplot
    ax.matshow(images[i])
    ax.axis('off')

# display the figure
fig.show()

```



```
[487]: from sklearn import metrics
print('Accuracy: {} \n'.format(metrics.accuracy_score(kmeansY,
↳ predicted_labels)))
```

Accuracy: 0.6486666666666666

[ ]:

# Kannada MNIST RF

May 2, 2021

## 1 Assignment 5: Principal Components Analysis & Clustering

You will compete in the Kaggle.com Kannada Digit Recognizer competition which involves classical digit recognition from hand-written images.

Read the competition rules, and download the MNIST training and test set data. This binary classification task is NOT what is required for the current assignment. In this assignment we are asking for a multiclass classifier. The entire MNIST data set will be used for input data. For this assignment, you will develop a classifier that may be used to predict which of the 10 digits is being written.

- (1) Begin by fitting a random forest classifier using the full set of 784 explanatory variables and the model training set (train.csv). Record the time it takes to fit the model and then evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (2) Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the 784 explanatory variables. Record the time it takes to identify the principal components.
- (3) Using the identified principal components from step (2), use the train.csv to build another random forest classifier. Record the time it takes to fit the model and to evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (4) Submit both the RF Classifier and the PCA RF Classifier to Kaggle.com, and report both scores along with your user name. I MUST have your user name to verify submission status.
- (5) The experiment we have proposed has a MAJOR design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com. Provide your Kaggle.com score and user ID.
- (6) Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels. (Follow the example here if needed: [kmeans mnist.pdf](#) download)

Report total elapsed time measures for the training set analysis. It is sufficient to run a single time-elapsed test for this assignment. In practice, we might consider the possibility of repeated executions of the relevant portions of the programs, much as the Benchmark Example programs do. Some code that might help you with reporting elapsed total time follows.

# Kannada MNIST PCA RF V2

May 2, 2021

## 1 Assignment 5: Principal Components Analysis & Clustering

You will compete in the Kaggle.com Kannada Digit Recognizer competition which involves classical digit recognition from hand-written images.

Read the competition rules, and download the MNIST training and test set data. This binary classification task is NOT what is required for the current assignment. In this assignment we are asking for a multiclass classifier. The entire MNIST data set will be used for input data. For this assignment, you will develop a classifier that may be used to predict which of the 10 digits is being written.

- (1) Begin by fitting a random forest classifier using the full set of 784 explanatory variables and the model training set (train.csv). Record the time it takes to fit the model and then evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (2) Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the 784 explanatory variables. Record the time it takes to identify the principal components.
- (3) Using the identified principal components from step (2), use the train.csv to build another random forest classifier. Record the time it takes to fit the model and to evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (4) Submit both the RF Classifier and the PCA RF Classifier to Kaggle.com, and report both scores along with your user name. I MUST have your user name to verify submission status.
- (5) The experiment we have proposed has a MAJOR design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com. Provide your Kaggle.com score and user ID.
- (6) Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels. (Follow the example here if needed: [kmeans mnist.pdf](#) download)

Report total elapsed time measures for the training set analysis. It is sufficient to run a single time-elapsed test for this assignment. In practice, we might consider the possibility of repeated executions of the relevant portions of the programs, much as the Benchmark Example programs do. Some code that might help you with reporting elapsed total time follows.

# Kannada Train Test Split

May 2, 2021

## 1 Assignment 5: Principal Components Analysis & Clustering

You will compete in the Kaggle.com Kannada Digit Recognizer competition which involves classical digit recognition from hand-written images.

Read the competition rules, and download the MNIST training and test set data. This binary classification task is NOT what is required for the current assignment. In this assignment we are asking for a multiclass classifier. The entire MNIST data set will be used for input data. For this assignment, you will develop a classifier that may be used to predict which of the 10 digits is being written.

- (1) Begin by fitting a random forest classifier using the full set of 784 explanatory variables and the model training set (train.csv). Record the time it takes to fit the model and then evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (2) Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the 784 explanatory variables. Record the time it takes to identify the principal components.
- (3) Using the identified principal components from step (2), use the train.csv to build another random forest classifier. Record the time it takes to fit the model and to evaluate the model on the test.csv data by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.
- (4) Submit both the RF Classifier and the PCA RF Classifier to Kaggle.com, and report both scores along with your user name. I MUST have your user name to verify submission status.
- (5) The experiment we have proposed has a MAJOR design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com. Provide your Kaggle.com score and user ID.
- (6) Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels. (Follow the example here if needed: [kmeans mnist.pdf](#) download)

Report total elapsed time measures for the training set analysis. It is sufficient to run a single time-elapsed test for this assignment. In practice, we might consider the possibility of repeated executions of the relevant portions of the programs, much as the Benchmark Example programs do. Some code that might help you with reporting elapsed total time follows.

```
start=datetime.now()
rf2.fit(trainimages,labels)
end=datetime.now()
print(end-start)
```

Relevant scikit-learn documentation includes:

Random Forest Classifier Metrics Classification Report Plot Digits Classification Sklearn Decomposition

(Optional reading) If you want to learn about the time it takes to execute individual functions or code segments within Python, this article demonstrates a variety of ways to do it

Regarding the F1 score and the evaluation of multiclass classifiers, refer to the literature on information retrieval. See pages 142–145 of this classic reference:

Manning, C. D., Raghaven, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge, UK: Cambridge University Press. [ISBN-13: 978-0521865715]

Or see pages 154–158 of the free online version of the book here

Additional information about this book is available online here

Programming Notes

One of the key parameters in setting up random forests is the number of explanatory variables to include in the individual trees. For this classification problem, I would suggest that we follow the advice of Müller and Guido (2017) and use `max_features = 'sqrt'`.

Regarding the other meta-parameters ... ensure that `bootstrap = True` and, given the large number of observations, we might as well keep the default value of `n_estimators = 10`.

Müller, A. C., & Guido, S. (2017). Introduction to machine learning with Python: A guide for data scientists. Sebastopol, CA: O'Reilly. [ISBN-13: 978-1449369415]. Code examples here

Another useful reference that discusses the MNIST data set and principal components analysis is:

VanderPlas, J. (2017). Python data science handbook: Essential tools for working with data. Sebastopol, CA: O'Reilly [ISBN-13: 978-1491912058].

Management Problem

From a management perspective, the predictive accuracy of models must be weighed against the costs of model development and implementation. Suppose you were the manager of a data science team responsible for implementing models for computer vision (classification of images analogous to the MNIST problem). Would you recommend using PCA as a preliminary to machine learning classification? Explain your thinking.

Grading Guidelines (50 points)

- (1) Data preparation, exploration, visualization (10 points)
- (2) Review research design and modeling methods (10 points)
- (3) Review results, evaluate models (10 points)

(4) Implementation and programming (10 points)

(5) Exposition, problem description, and management recommendations (10 points)

#### Deliverables and File Formats

Provide a double-spaced paper with a two-page maximum for the text. The paper should include all a discussion of all graded elements but focus particularly on insights. Include your Python code and output as an appendix. Upload this as a single .pdf file.

#### Formatting Python Code

Refer to Google's Python Style Guide for ideas about formatting Python code

Comment often and in detail, highlighting major sections of code, describing the thinking behind the modeling and programming methods being employed.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

```
[ ]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6

import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

[ ]: df_train = pd.read_csv('/kaggle/input/Kannada-MNIST/train.csv')
df_test = pd.read_csv('/kaggle/input/Kannada-MNIST/test.csv')

[ ]: df_train.head()

[ ]: df_train['label'].dtype

[ ]: type(df_train['label'])

[ ]: df_train['label'].unique()

[ ]: df_test.head()

[ ]: print(df_train.shape)
print(df_test.shape)
```



```
[ ]: num = 4
      plot_num = df_train.iloc[num, 1:]
      plot_num = np.array(plot_num).reshape(28, -1)
      plt.imshow(plot_num, cmap='gray')
      plt.title(f'Label: {df_train.iloc[num, 0]}')
      plt.show()
```

```
[ ]: df_train.values
```

```
[ ]: df_test.values
```

```
[ ]: df_train.values.shape
```

```
[ ]: df_test.values.shape
```

```
[ ]: X = df_train.iloc[:, 1:].values
      X
```

```
[ ]: X.shape
```

```
[ ]: y = df_train.iloc[:, 0].values
      y
```

```
[ ]: y.shape
```

```
[ ]: X_test = df_test.iloc[:, 1:].values
      X_test
```

```
[ ]: X_test.shape
```

## 2 Step 5 - Rerun the experiment in a way that is consistent with a training-

### 3 and-test regimen

```
[ ]: from sklearn.model_selection import train_test_split
      # test_size: what proportion of original data is used for test set
      X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=1/7.0,
      ↪random_state=0)
      print(X_train.shape)
      print(X_val.shape)
```

```
[ ]: def algorithm_pipeline(X_train_data, X_test_data, y_train_data, y_test_data,
      model, param_grid, cv=10, scoring_fit='f1_micro'):
      gs = GridSearchCV(
          estimator=model,
```

```

        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit,
        verbose=2,
        return_train_score=True # set this for train score
    )

    fitted_model = gs.fit(X_train_data, y_train_data.ravel())
    pred = fitted_model.predict(X_test_data)

    return gs, fitted_model, pred

```

```

[ ]: import xgboost as xgb

xgb_class_model = xgb.XGBRFClassifier()

param_grid = {
    'n_estimators': [10, 25, 50],
    'colsample_bytree': [0.6],
    'max_depth': [6, 10, 15],
    'subsample': [0.6],
    'tree_method': ['gpu_hist'],
}

%time gs, xgb_class_model, pred = algorithm_pipeline(X_train, X_val, y_train, \
    ↪ y_val, xgb_class_model, \
                                           param_grid, cv=3)

```

```

[ ]: print(xgb_class_model.best_params_)

```

```

[ ]: df = pd.DataFrame(gs.cv_results_)
df

```

```

[ ]: results = ['mean_test_score',
               'mean_train_score',
               'std_test_score',
               'std_train_score']

def pooled_var(stds):
    # https://en.wikipedia.org/wiki/Pooled_variance#Pooled_standard_deviation
    n = 5 # size of each group
    return np.sqrt(sum((n-1)*(stds**2))/ len(stds)*(n-1))

fig, axes = plt.subplots(1, len(param_grid),
                        figsize = (5*len(param_grid), 7),
                        sharey='row')

```

```

axes[0].set_ylabel("Score", fontsize=25)

for idx, (param_name, param_range) in enumerate(param_grid.items()):
    grouped_df = df.groupby(f'param_{param_name}')[results]\
        .agg({'mean_train_score': 'mean',
              'mean_test_score': 'mean',
              'std_train_score': pooled_var,
              'std_test_score': pooled_var})

    previous_group = df.groupby(f'param_{param_name}')[results]
    axes[idx].set_xlabel(param_name, fontsize=30)
    axes[idx].set_ylim(0.0, 1.1)
    lw = 2
    axes[idx].plot(param_range, grouped_df['mean_train_score'], label="Training\
→score",
                   color="darkorange", lw=lw)
    axes[idx].fill_between(param_range, grouped_df['mean_train_score'] -\
→grouped_df['std_train_score'],
                           grouped_df['mean_train_score'] +\
→grouped_df['std_train_score'], alpha=0.2,
                           color="darkorange", lw=lw)
    axes[idx].plot(param_range, grouped_df['mean_test_score'],\
→label="Cross-validation score",
                   color="navy", lw=lw)
    axes[idx].fill_between(param_range, grouped_df['mean_test_score'] -\
→grouped_df['std_test_score'],
                           grouped_df['mean_test_score'] +\
→grouped_df['std_test_score'], alpha=0.2,
                           color="navy", lw=lw)

handles, labels = axes[0].get_legend_handles_labels()
fig.suptitle('Validation curves', fontsize=40)
fig.legend(handles, labels, loc=8, ncol=2, fontsize=20)

fig.subplots_adjust(bottom=0.25, top=0.85)
plt.show()

```

```

[ ]: from xgboost import XGBRFClassifier
xgb_rf_class = XGBRFClassifier(colsample_bytree=0.6, max_depth=15,\
→n_estimators=50, subsample=0.6)

%time xgb_rf_class.fit(X_train, y_train.ravel())

[ ]: %time y_pred_val = xgb_rf_class.predict(X_val)
y_pred_val

```

```
[ ]: %time y_pred_train = xgb_rf_class.predict(X_train)
y_pred_train
```

```
[ ]: from sklearn.metrics import confusion_matrix

confusion_matrix(y_val, y_pred_val)
```

```
[ ]: confusion_matrix(y_train, y_pred_train)
```

```
[ ]: from sklearn.metrics import f1_score

print(f1_score(y_val, y_pred_val, average='micro'))
```

## 4 Steps 5 Submission

```
[ ]: y_pred_test = xgb_rf_class.predict(X_test)
```

```
[ ]: submission_data = np.c_[df_test["id"].values, y_pred_test]
submission_df = pd.DataFrame(data = submission_data, columns = ["id", "label"])
submission_df['id'] = submission_df['id'].astype('int64')
submission_df['label'] = submission_df['label'].astype('int64')
submission_df
```

```
[ ]: submission_df.to_csv("submission.csv", index = False)
```

```
[ ]:
```

```
start=datetime.now()
rf2.fit(trainimages,labels)
end=datetime.now()
print(end-start)
```

Relevant scikit-learn documentation includes:

Random Forest Classifier Metrics Classification Report Plot Digits Classification Sklearn Decomposition

(Optional reading) If you want to learn about the time it takes to execute individual functions or code segments within Python, this article demonstrates a variety of ways to do it

Regarding the F1 score and the evaluation of multiclass classifiers, refer to the literature on information retrieval. See pages 142–145 of this classic reference:

Manning, C. D., Raghaven, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge, UK: Cambridge University Press. [ISBN-13: 978-0521865715]

Or see pages 154–158 of the free online version of the book here

Additional information about this book is available online here

Programming Notes

One of the key parameters in setting up random forests is the number of explanatory variables to include in the individual trees. For this classification problem, I would suggest that we follow the advice of Müller and Guido (2017) and use `max_features = 'sqrt'`.

Regarding the other meta-parameters ... ensure that `bootstrap = True` and, given the large number of observations, we might as well keep the default value of `n_estimators = 10`.

Müller, A. C., & Guido, S. (2017). Introduction to machine learning with Python: A guide for data scientists. Sebastopol, CA: O'Reilly. [ISBN-13: 978-1449369415]. Code examples here

Another useful reference that discusses the MNIST data set and principal components analysis is:

VanderPlas, J. (2017). Python data science handbook: Essential tools for working with data. Sebastopol, CA: O'Reilly [ISBN-13: 978-1491912058].

Management Problem

From a management perspective, the predictive accuracy of models must be weighed against the costs of model development and implementation. Suppose you were the manager of a data science team responsible for implementing models for computer vision (classification of images analogous to the MNIST problem). Would you recommend using PCA as a preliminary to machine learning classification? Explain your thinking.

Grading Guidelines (50 points)

- (1) Data preparation, exploration, visualization (10 points)
- (2) Review research design and modeling methods (10 points)
- (3) Review results, evaluate models (10 points)

(4) Implementation and programming (10 points)

(5) Exposition, problem description, and management recommendations (10 points)

#### Deliverables and File Formats

Provide a double-spaced paper with a two-page maximum for the text. The paper should include all a discussion of all graded elements but focus particularly on insights. Include your Python code and output as an appendix. Upload this as a single .pdf file.

#### Formatting Python Code

Refer to Google's Python Style Guide for ideas about formatting Python code

Comment often and in detail, highlighting major sections of code, describing the thinking behind the modeling and programming methods being employed.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

```
[ ]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6

import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

[ ]: df_train = pd.read_csv('/kaggle/input/Kannada-MNIST/train.csv')
df_test = pd.read_csv('/kaggle/input/Kannada-MNIST/test.csv')

[ ]: df_train.head()

[ ]: df_train['label'].dtype

[ ]: type(df_train['label'])

[ ]: df_train['label'].unique()

[ ]: df_test.head()

[ ]: print(df_train.shape)
print(df_test.shape)
```

```
[ ]: num = 4
      plot_num = df_train.iloc[num, 1:]
      plot_num = np.array(plot_num).reshape(28, -1)
      plt.imshow(plot_num, cmap='gray')
      plt.title(f'Label: {df_train.iloc[num, 0]}')
      plt.show()
```

```
[ ]: df_train.values
```

```
[ ]: df_test.values
```

```
[ ]: df_train.values.shape
```

```
[ ]: df_test.values.shape
```

```
[ ]: X = df_train.iloc[:, 1:].values
      X
```

```
[ ]: X.shape
```

```
[ ]: y = df_train.iloc[:, 0].values
      y
```

```
[ ]: y.shape
```

```
[ ]: X_test = df_test.iloc[:, 1:].values
      X_test
```

```
[ ]: X_test.shape
```

## 2 Step 2 - Principal Component Analysis for the combined training and test

### 3 data

#### 3.1 Standardize the Data

```
[ ]: from sklearn.preprocessing import StandardScaler

      scaler = StandardScaler()

      # Fit on training set only.
      scaler.fit(X)

      # Apply transform to both the training set and the test set.
      train_img = scaler.transform(X)
      test_img = scaler.transform(X_test)
```

### 3.2 Import and Apply PCA

```
[ ]: from sklearn.decomposition import PCA

# Make an instance of the Model
pca = PCA(.95)

[ ]: pca.fit(train_img)

[ ]: pca_components = pca.n_components_
pca_components

[ ]: import plotly.express as px
from sklearn.decomposition import PCA

pca_scatter = PCA(n_components=2)
components = pca_scatter.fit_transform(df_train)

principalDf = pd.DataFrame(data = components, columns = ['PC 1', 'PC 2'])
finalDf = pd.concat([principalDf, df_train[['label']]], axis = 1)

[ ]: fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)

ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
colors = ['red', 'green', 'blue', 'orange', 'purple', 'brown', 'navy', 'magenta', 'teal', 'black']

for target, color in zip(targets, colors):
    indicesToKeep = finalDf['label'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'PC 1'],
               finalDf.loc[indicesToKeep, 'PC 2'],
               c = color,
               s = 50)
ax.legend(targets)
ax.grid()
```

4 Step 3 - Using the identified principal components from step 2, use the



## 5 train.csv to build another random forest classifier.

```
[ ]: # Apply transform to the training set
train_img = pca.transform(train_img)

[ ]: train_img.shape

[ ]: test_img = pca.transform(test_img)

[ ]: test_img.shape

[ ]: import xgboost as xgb
from xgboost import XGBRFClassifier

xgb_rf_model = XGBRFClassifier()

param_grid = {
    'eta': [0.05],
    'n_estimators': [500, 1000],
    'colsample_bytree': [0.7, 0.8],
    'max_depth': [6, 10],
    'reg_alpha': [0, 1],
    'reg_lambda': [1, 2],
    'subsample': [0.5, 0.9]
}

[ ]: xgb_rf_train_img = XGBRFClassifier(n_estimators=50, learning_rate=0.05,
    ↪subsample=0.7, colsample_bytree=0.7,
    max_depth=6, reg_alpha=0, eval_metric='mlogloss')

%time xgb_rf_train_img.fit(train_img, y)

[ ]: %time y_pred_train_img = xgb_rf_train_img.predict(train_img)
y_pred_train_img

[ ]: from sklearn.metrics import confusion_matrix

confusion_matrix(y, y_pred_train_img)

[ ]: from sklearn.metrics import f1_score
print(f1_score(y, y_pred_train_img, average='micro'))

[ ]: import pandas as pd
import numpy as np
from scipy import interp

from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import roc_curve, auc
```

```

from sklearn.preprocessing import LabelBinarizer

def class_report(y_true, y_pred, y_score=None, average='micro'):
    if y_true.shape != y_pred.shape:
        print("Error! y_true %s is not the same shape as y_pred %s" % (
            y_true.shape,
            y_pred.shape)
        )
        return

    lb = LabelBinarizer()

    if len(y_true.shape) == 1:
        lb.fit(y_true)

    #Value counts of predictions
    labels, cnt = np.unique(
        y_pred,
        return_counts=True)
    n_classes = len(labels)
    pred_cnt = pd.Series(cnt, index=labels)

    metrics_summary = precision_recall_fscore_support(
        y_true=y_true,
        y_pred=y_pred,
        labels=labels)

    avg = list(precision_recall_fscore_support(
        y_true=y_true,
        y_pred=y_pred,
        average='weighted'))

    metrics_sum_index = ['precision', 'recall', 'f1-score', 'support']
    class_report_df = pd.DataFrame(
        list(metrics_summary),
        index=metrics_sum_index,
        columns=labels)

    support = class_report_df.loc['support']
    total = support.sum()
    class_report_df['avg / total'] = avg[:-1] + [total]

    class_report_df = class_report_df.T
    class_report_df['pred'] = pred_cnt
    class_report_df['pred'].iloc[-1] = total

    if not (y_score is None):

```

```

fpr = dict()
tpr = dict()
roc_auc = dict()
for label_it, label in enumerate(labels):
    fpr[label], tpr[label], _ = roc_curve(
        (y_true == label).astype(int),
        y_score[:, label_it])

    roc_auc[label] = auc(fpr[label], tpr[label])

if average == 'micro':
    if n_classes <= 2:
        fpr["avg / total"], tpr["avg / total"], _ = roc_curve(
            lb.transform(y_true).ravel(),
            y_score[:, 1].ravel())
    else:
        fpr["avg / total"], tpr["avg / total"], _ = roc_curve(
            lb.transform(y_true).ravel(),
            y_score.ravel())

    roc_auc["avg / total"] = auc(
        fpr["avg / total"],
        tpr["avg / total"])

elif average == 'macro':
    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([
        fpr[i] for i in labels]
    ))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in labels:
        mean_tpr += interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr

    roc_auc["avg / total"] = auc(fpr["macro"], tpr["macro"])

class_report_df['AUC'] = pd.Series(roc_auc)

return class_report_df

```

```
[ ]: report_with_auc = class_report(  
    y_true=y.ravel(),  
    y_pred=xgb_rf_train_img.predict(train_img),  
    y_score=xgb_rf_train_img.predict_proba(train_img))  
  
print(report_with_auc)
```

## 6 Steps 2 and 3 Submission

```
[ ]: y_pred_test = xgb_rf_train_img.predict(test_img)
```

```
[ ]: submission_data = np.c_[df_test["id"].values, y_pred_test]  
submission_df = pd.DataFrame(data = submission_data, columns = ["id", "label"])  
submission_df['id'] = submission_df['id'].astype('int64')  
submission_df['label'] = submission_df['label'].astype('int64')  
submission_df
```

```
[ ]: submission_df.to_csv("submission.csv", index = False)
```

```
[ ]:
```

```
start=datetime.now()
rf2.fit(trainimages,labels)
end=datetime.now()
print(end-start)
```

Relevant scikit-learn documentation includes:

Random Forest Classifier Metrics Classification Report Plot Digits Classification Sklearn Decomposition

(Optional reading) If you want to learn about the time it takes to execute individual functions or code segments within Python, this article demonstrates a variety of ways to do it

Regarding the F1 score and the evaluation of multiclass classifiers, refer to the literature on information retrieval. See pages 142–145 of this classic reference:

Manning, C. D., Raghaven, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge, UK: Cambridge University Press. [ISBN-13: 978-0521865715]

Or see pages 154–158 of the free online version of the book here

Additional information about this book is available online here

Programming Notes

One of the key parameters in setting up random forests is the number of explanatory variables to include in the individual trees. For this classification problem, I would suggest that we follow the advice of Müller and Guido (2017) and use `max_features = 'sqrt'`.

Regarding the other meta-parameters ... ensure that `bootstrap = True` and, given the large number of observations, we might as well keep the default value of `n_estimators = 10`.

Müller, A. C., & Guido, S. (2017). Introduction to machine learning with Python: A guide for data scientists. Sebastopol, CA: O'Reilly. [ISBN-13: 978-1449369415]. Code examples here

Another useful reference that discusses the MNIST data set and principal components analysis is:

VanderPlas, J. (2017). Python data science handbook: Essential tools for working with data. Sebastopol, CA: O'Reilly [ISBN-13: 978-1491912058].

Management Problem

From a management perspective, the predictive accuracy of models must be weighed against the costs of model development and implementation. Suppose you were the manager of a data science team responsible for implementing models for computer vision (classification of images analogous to the MNIST problem). Would you recommend using PCA as a preliminary to machine learning classification? Explain your thinking.

Grading Guidelines (50 points)

- (1) Data preparation, exploration, visualization (10 points)
- (2) Review research design and modeling methods (10 points)
- (3) Review results, evaluate models (10 points)

(4) Implementation and programming (10 points)

(5) Exposition, problem description, and management recommendations (10 points)

#### Deliverables and File Formats

Provide a double-spaced paper with a two-page maximum for the text. The paper should include all a discussion of all graded elements but focus particularly on insights. Include your Python code and output as an appendix. Upload this as a single .pdf file.

#### Formatting Python Code

Refer to Google's Python Style Guide for ideas about formatting Python code

Comment often and in detail, highlighting major sections of code, describing the thinking behind the modeling and programming methods being employed.

For all Kaggle competitions, you must submit a screen snapshot that identifies you along with your scores on the submissions. Submit your work as a single .pdf file that is legible. Include your code as an appendix. Look at the rubric to see how you will be graded. Your work will be compared against your peers on the performance metric(s).

```
[ ]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6

import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

[ ]: df_train = pd.read_csv('/kaggle/input/Kannada-MNIST/train.csv')
df_test = pd.read_csv('/kaggle/input/Kannada-MNIST/test.csv')

[ ]: df_train.head()

[ ]: df_train['label'].dtype

[ ]: type(df_train['label'])

[ ]: df_train['label'].unique()

[ ]: df_test.head()

[ ]: print(df_train.shape)
print(df_test.shape)
```

```
[ ]: num = 4
plot_num = df_train.iloc[num, 1:]
plot_num = np.array(plot_num).reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.title(f'Label: {df_train.iloc[num, 0]}')
plt.show()
```

## 2 Step 1 - Random Forest Classifier

```
[ ]: df_train.values
```

```
[ ]: df_test.values
```

```
[ ]: df_train.values.shape
```

```
[ ]: df_test.values.shape
```

```
[ ]: X = df_train.iloc[:, 1:].values
X
```

```
[ ]: X.shape
```

```
[ ]: y = df_train.iloc[:, 0].values
y
```

```
[ ]: y.shape
```

```
[ ]: import xgboost as xgb
from xgboost import XGBRFClassifier

param_grid = {
    'eta': [0.05],
    'n_estimators': [500, 1000],
    'colsample_bytree': [0.7, 0.8],
    'max_depth': [6, 10],
    'reg_alpha': [0, 1],
    'reg_lambda': [1, 2],
    'subsample': [0.5, 0.9]
}
```

```
[ ]: xgb_rf = XGBRFClassifier(n_estimators=50, learning_rate=0.05, subsample=0.7,
    ↪ colsample_bytree=0.7,
    max_depth=6, reg_alpha=0, eval_metric='mlogloss')

%time xgb_rf.fit(X, y)
```

```
[ ]: %time y_pred = xgb_rf.predict(X)
y_pred
```

```
[ ]: from sklearn.metrics import confusion_matrix

confusion_matrix(y, y_pred)
```

```
[ ]: from sklearn.metrics import f1_score
print(f1_score(y, y_pred, average='micro'))
```

```
[ ]: import pandas as pd
import numpy as np
from scipy import interp

from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import LabelBinarizer

def class_report(y_true, y_pred, y_score=None, average='micro'):
    if y_true.shape != y_pred.shape:
        print("Error! y_true %s is not the same shape as y_pred %s" % (
            y_true.shape,
            y_pred.shape)
        )
        return

    lb = LabelBinarizer()

    if len(y_true.shape) == 1:
        lb.fit(y_true)

    #Value counts of predictions
    labels, cnt = np.unique(
        y_pred,
        return_counts=True)
    n_classes = len(labels)
    pred_cnt = pd.Series(cnt, index=labels)

    metrics_summary = precision_recall_fscore_support(
        y_true=y_true,
        y_pred=y_pred,
        labels=labels)

    avg = list(precision_recall_fscore_support(
        y_true=y_true,
        y_pred=y_pred,
        average='weighted'))
```



```

metrics_sum_index = ['precision', 'recall', 'f1-score', 'support']
class_report_df = pd.DataFrame(
    list(metrics_summary),
    index=metrics_sum_index,
    columns=labels)

support = class_report_df.loc['support']
total = support.sum()
class_report_df['avg / total'] = avg[:-1] + [total]

class_report_df = class_report_df.T
class_report_df['pred'] = pred_cnt
class_report_df['pred'].iloc[-1] = total

if not (y_score is None):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for label_it, label in enumerate(labels):
        fpr[label], tpr[label], _ = roc_curve(
            (y_true == label).astype(int),
            y_score[:, label_it])

        roc_auc[label] = auc(fpr[label], tpr[label])

    if average == 'micro':
        if n_classes <= 2:
            fpr["avg / total"], tpr["avg / total"], _ = roc_curve(
                lb.transform(y_true).ravel(),
                y_score[:, 1].ravel())
        else:
            fpr["avg / total"], tpr["avg / total"], _ = roc_curve(
                lb.transform(y_true).ravel(),
                y_score.ravel())

        roc_auc["avg / total"] = auc(
            fpr["avg / total"],
            tpr["avg / total"])

    elif average == 'macro':
        # First aggregate all false positive rates
        all_fpr = np.unique(np.concatenate([
            fpr[i] for i in labels
        ]))

        # Then interpolate all ROC curves at this points

```

```

mean_tpr = np.zeros_like(all_fpr)
for i in labels:
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr

roc_auc["avg / total"] = auc(fpr["macro"], tpr["macro"])

class_report_df['AUC'] = pd.Series(roc_auc)

return class_report_df

```

```

[ ]: report_with_auc = class_report(
    y_true=y.ravel(),
    y_pred=xgb_rf.predict(X),
    y_score=xgb_rf.predict_proba(X))

print(report_with_auc)

```

### 3 Step 1 Submission

```

[ ]: X_test = df_test.iloc[:, 1:].values
X_test

```

```

[ ]: y_pred_test = xgb_rf.predict(X_test)

```

```

[ ]: submission_data = np.c_[df_test["id"].values, y_pred_test]
submission_df = pd.DataFrame(data = submission_data, columns = ["id", "label"])
submission_df['id'] = submission_df['id'].astype('int64')
submission_df['label'] = submission_df['label'].astype('int64')
submission_df

```

```

[ ]: submission_df.to_csv("submission.csv", index = False)

```