devnagari-handwritten-chars-classification-scikit

November 10, 2021

1 Handwritten Devnagari Character classification

1.0.1 Author: Sanjay Jaras

1.0.2 Bellevue University

1.1 Import Libraries

[ ]:

```
import os
from IPython.display import clear_output
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
import random
import PIL
from PIL import Image
from skimage.transform import resize
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from xgboost.sklearn import XGBClassifier
import seaborn as sns
from sklearn import svm
```

1.2 Configurations

[ ]:

```
random_state = 17
np.random.seed(random_state)
import warnings
warnings.filterwarnings('ignore')
```

1.3 Define datasource paths

[ ]:
```
base_path = "DevanagariHandwrittenCharacterDataset"
#base_path = "../input/devnagrihandwrittenchars/
 ↪DevanagariHandwrittenCharacterDataset"
train_path = os.path.join(base_path, "Train")
test_path = os.path.join(base_path, "Test")
```

1.4 Function to scan the folders and load images in array.

[ ]:
```
def load_image_to_array(file_path):
    with open(file_path, "rb") as f:
        img = PIL.Image.open(f)
        nparr = np.asarray(img)
        # plt.imshow(nparr)
        #nparr = nparr[:, :, np.newaxis]
        return nparr


def read_data_from_folder(folder_path, read_first_record_only=False):
    imgs = []
    labels = []
    for folder in tqdm(os.listdir(folder_path)):
        sub_folder = os.path.join(folder_path, folder)
        for f in os.listdir(sub_folder):
            img = load_image_to_array(os.path.join(sub_folder, f))
            imgs.append(img)
            labels.append(folder)
            if read_first_record_only:
                break
    return np.asarray(imgs), np.asarray(labels)
```

1.5 Sample images from all source folders

[ ]:
```
sample_imgs, sample_labels = read_data_from_folder(train_path, True)
sample_imgs.shape
```

```
100%|      | 46/46 [00:00<00:00, 917.22it/s]
```

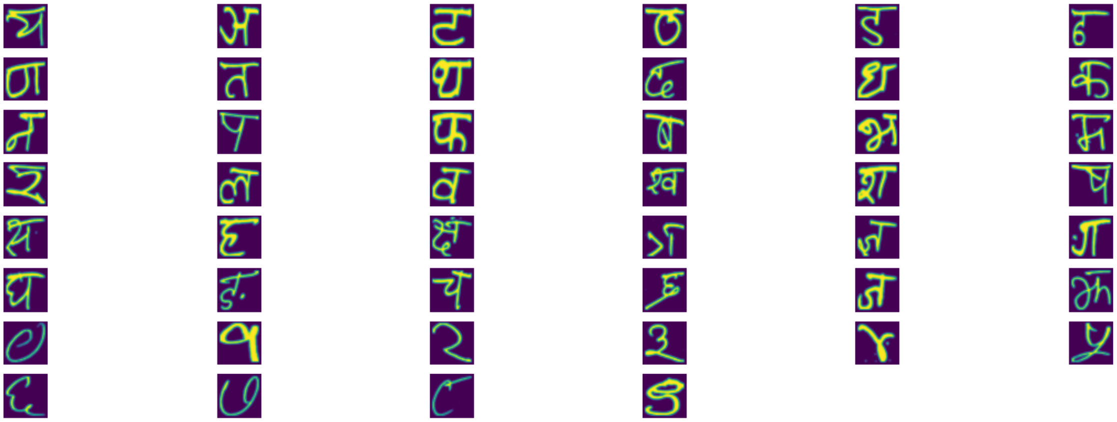
[ ]:
```
(46, 32, 32)
```

1.6 Function to Display Images

[ ]:
```
def display_image(imgarr):
    plt.figure(figsize=(20, 40))
    for i in range(len(imgarr)):
        plt.subplot(46, 6, i+1)
        img = resize(imgarr[i], [100, 100])
        plt.imshow(img)
        plt.axis('off')
    plt.show()
```

1.7 Show one sample image from each of input training folder

```
[ ]: display_image(sample_imgs)
```



1.8 Load Training and Test Dataset

```
[ ]: print("Loading training data....")
     train_data_img, train_data_labels = read_data_from_folder(train_path)
     print("Loading test data....")
     test_data_imgs, test_data_labels = read_data_from_folder(test_path)
```

```
Loading training data…

100%|      | 46/46 [00:11<00:00,  4.03it/s]

Loading test data…

100%|      | 46/46 [00:02<00:00, 21.82it/s]
```

1.9 Display Dataset shapes

```
[ ]: print("Training data imgs shape", train_data_img.shape)
     print("Training data labels shape", train_data_labels.shape)
     print("Test data imgs shape", test_data_imgs.shape)
     print("Test data labels shape", test_data_labels.shape)
```

```
Training data imgs shape (78200, 32, 32)
Training data labels shape (78200,)
Test data imgs shape (13800, 32, 32)
Test data labels shape (13800,)
```

1.10 Reshape dataset to use with models

[ ]:

```
train_data_img = train_data_img.reshape(train_data_img.shape[0], train_data_img.
 ↪shape[1] * train_data_img.shape[2])
test_data_imgs = test_data_imgs.reshape(test_data_imgs.shape[0], test_data_imgs.
 ↪shape[1] * test_data_imgs.shape[2])
```

[ ]:

```
print("Training data imgs shape", train_data_img.shape)
print("Training data labels shape", train_data_labels.shape)
print("Test data imgs shape", test_data_imgs.shape)
print("Test data labels shape", test_data_labels.shape)
```

```
Training data imgs shape (78200, 1024)
Training data labels shape (78200,)
Test data imgs shape (13800, 1024)
Test data labels shape (13800,)
```

1.11 Add column for label in dataframe

[ ]:

```
train_df = pd.DataFrame(train_data_img)
train_df["label"] = train_data_labels
```

[ ]:

```
train_df
```

[ ]:

```
       0  1  2  3  4  5  6  7  8  9  …  1015  1016  1017  1018  1019  1020  \
0      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
1      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
2      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
3      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
4      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
…     .. .. .. .. .. .. .. .. .. ..  …   …    …    …    …    …    …
78195  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
78196  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
78197  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
78198  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
78199  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0

       1021  1022  1023             label
0         0     0     0  character_26_yaw
1         0     0     0  character_26_yaw
2         0     0     0  character_26_yaw
3         0     0     0  character_26_yaw
4         0     0     0  character_26_yaw
…       …    …    …                …
78195     0     0     0           digit_9
78196     0     0     0           digit_9
78197     0     0     0           digit_9
78198     0     0     0           digit_9
```

```
78199     0     0     0           digit_9

[78200 rows x 1025 columns]
```

[ ]:

```
test_df = pd.DataFrame(test_data_imgs)
test_df["label"] = test_data_labels
```

[ ]:

```
test_df
```

[ ]:

```
       0  1  2  3  4  5  6  7  8  9  …  1015  1016  1017  1018  1019  1020  \
0      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
1      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
2      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
3      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
4      0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
…     .. .. .. .. .. .. .. .. .. ..  …   …     …    …     …     …     …
13795  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
13796  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
13797  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
13798  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0
13799  0  0  0  0  0  0  0  0  0  0  …     0     0     0     0     0     0

       1021  1022  1023             label
0         0     0     0  character_26_yaw
1         0     0     0  character_26_yaw
2         0     0     0  character_26_yaw
3         0     0     0  character_26_yaw
4         0     0     0  character_26_yaw
…       …     …     …                 …
13795     0     0     0           digit_9
13796     0     0     0           digit_9
13797     0     0     0           digit_9
13798     0     0     0           digit_9
13799     0     0     0           digit_9

[13800 rows x 1025 columns]
```

1.12 Class to normalize

[ ]:

```
class Normalizer(BaseEstimator, TransformerMixin):
    def __init__(self, labelCol ):
        self.labelCol = labelCol

    def transform(self, df):
        for attr in  tqdm(df.columns):
            if attr != self.labelCol:
                df[attr] = df[attr]/255.0
        return df

    def fit(self, X, y = None):
        return self
```

[ ]:

```
normalizer = Normalizer("label")
train_new = normalizer.transform(train_df)
#train_new[122]
```

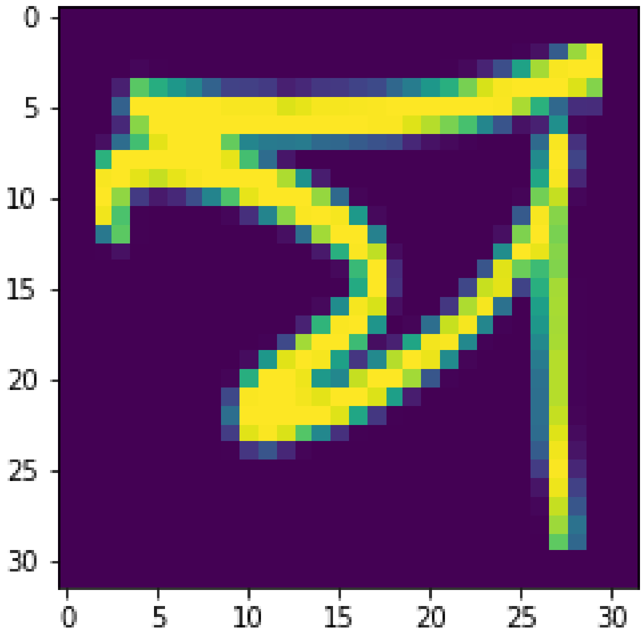
```
100%|      | 1025/1025 [00:05<00:00, 171.60it/s]
```

[ ]:

```
a = np.array(train_df.iloc[0][:1024], dtype='float')
a = a.reshape(32,32)
plt.imshow(a)
```

[ ]:

```
<matplotlib.image.AxesImage at 0x7f42c0197a30>
```



1.13 Class for label encoding

[ ]:

```
class MyLabelEncoder(LabelEncoder):
    def __init__(self, labelCol ):
        self.labelCol = labelCol

    def transform(self, df):
        df[self.labelCol] = LabelEncoder.transform(self, df[self.labelCol])
        return df
    def fit(self, X, y = None):
        LabelEncoder.fit(self, X[self.labelCol])
        return self
```

1.14 classes Available

[ ]:

```
labelEncoder = MyLabelEncoder("label")
labelEncoder.fit(train_df)

labelEncoder.classes_
# array([1, 2, 6])
#labelEncoder.transform(['character_10_yna', 'character_11_taamatar',␣
 ↪'character_12_thaa','character_13_daa'])
# array([0, 0, 1, 2]...)
# >>> le.inverse_transform([0, 0, 1, 2])
# array([1, 1, 2, 6])
```

[ ]:

```
array(['character_10_yna', 'character_11_taamatar', 'character_12_thaa',
       'character_13_daa', 'character_14_dhaa', 'character_15_adna',
       'character_16_tabala', 'character_17_tha', 'character_18_da',
       'character_19_dha', 'character_1_ka', 'character_20_na',
       'character_21_pa', 'character_22_pha', 'character_23_ba',
       'character_24_bha', 'character_25_ma', 'character_26_yaw',
       'character_27_ra', 'character_28_la', 'character_29_waw',
       'character_2_kha', 'character_30_motosaw',
       'character_31_petchiryakha', 'character_32_patalosaw',
       'character_33_ha', 'character_34_chhya', 'character_35_tra',
       'character_36_gya', 'character_3_ga', 'character_4_gha',
       'character_5_kna', 'character_6_cha', 'character_7_chha',
       'character_8_ja', 'character_9_jha', 'digit_0', 'digit_1',
       'digit_2', 'digit_3', 'digit_4', 'digit_5', 'digit_6', 'digit_7',
       'digit_8', 'digit_9'], dtype=object)
```

1.15 Create pipeline

[ ]:

```
transform_pipeline = Pipeline([("Normalizer",  normalizer), ("label-encode",␣
 ↪labelEncoder)])
```

1.16 Apply transformation pipeline

[ ]:

```
train_df = transform_pipeline.transform(train_df)

test_df = transform_pipeline.transform(test_df)
```

```
100%|          | 1025/1025 [00:00<00:00, 5337.03it/s]
100%|          | 1025/1025 [00:00<00:00, 1246.53it/s]
```

1.17 Split input and labels in separate dataframes

[ ]:

```
train_X = train_df.loc[:, train_df.columns != "label"]
train_Y = train_df["label"]
test_X = test_df.loc[:, test_df.columns != "label"]
test_Y = test_df["label"]
```

[ ]:

```
cv = RepeatedStratifiedKFold(n_splits=2, n_repeats=1, random_state=random_state)
```

1.18 Create sample dataset for grid search

[ ]:

```
#import sklearn
#sklearn.metrics.SCORERS.keys()
train_x_grid = train_X[0:1]
train_y_grid = train_Y[0:1]
cnt = 100
num_rec_per_class = 1700
for i in range(0, train_X.shape[0]//46):
    #print(i*num_rec_per_class, i*num_rec_per_class+cnt)
    train_x_grid = np.append(train_x_grid, train_X[i*num_rec_per_class:
 ↪i*num_rec_per_class+cnt], axis=0)
    train_y_grid = np.append(train_y_grid, train_Y[i*num_rec_per_class:
 ↪i*num_rec_per_class+cnt], axis=0)
    #train_x_grid.append(train_X[i*num_rec_per_class:i*num_rec_per_class+cnt])
print(np.array(train_x_grid).shape)
print(np.array(train_y_grid).shape)
```

```
(4601, 1024)
(4601,)
```

1.19 Grid Search

[ ]:

```
param_grid = [
    {'n_neighbors': [5, 20, 50], 'leaf_size': [10, 20, 30]},
    {'min_weight_fraction_leaf': [0, 0.2, 0.5]},
    {'n_estimators': [100, 200, 300]},
    #{'learning_rate': [0.1, 0.3], 'max_depth':[10, 30], 'n_estimators':[50,␣
 ↪100, 150]},
    {'learning_rate': [0.1, 0.3], 'max_depth':[
        10, 30, -1], 'n_estimators':[50, 100, 150]},
    # {'learning_rate': [0.1, 0.3], 'max_depth':[10, 30, -1], 'n_estimators':
 ↪[50, 100, 150]},
    {'degree': [3, 10, 20], 'decision_function_shape': ['ovo', 'ovr']}

]
estimators = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(random_state=random_state),
    RandomForestClassifier(random_state=random_state),
    #GradientBoostingClassifier(random_state= random_state),
    LGBMClassifier(random_state=random_state),
    #XGBClassifier(random_state= random_state,␣
 ↪use_label_encoder=False),random_state
    svm.SVC(random_state=random_state)
]
# 'kernel': ['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
grid_result = []
for i in tqdm(range(len(estimators))):
    print("*" * 100)
    print("Evaluating Model", estimators[i].__class__.__name__)
    grid_reg = GridSearchCV(
        estimators[i], param_grid=param_grid[i], cv=cv, verbose=10, n_jobs=12,␣
 ↪scoring="f1_macro")
    grid_reg.fit(train_x_grid, train_y_grid)
    print("Best Score:", grid_reg.best_score_)
    print("Best Params:", grid_reg.best_params_)
    grid_result.append(grid_reg)
```

```
  0%|          | 0/5 [00:00<?, ?it/s]

********************************************************************************
********************
Evaluating Model KNeighborsClassifier
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV 1/2; 2/9] START leaf_size=10, n_neighbors=20…
[CV 2/2; 1/9] START leaf_size=10,
n_neighbors=5…[CV 1/2; 1/9] START leaf_size=10,
n_neighbors=5…

[CV 2/2; 2/9] START leaf_size=10, n_neighbors=20…
[CV 1/2; 3/9] START leaf_size=10, n_neighbors=50…
[CV 2/2; 3/9] START leaf_size=10, n_neighbors=50…
[CV 1/2; 6/9] START leaf_size=20, n_neighbors=50…
[CV 1/2; 4/9] START leaf_size=20, n_neighbors=5…
[CV 2/2; 4/9] START leaf_size=20, n_neighbors=5…
[CV 1/2; 5/9] START leaf_size=20, n_neighbors=20…
[CV 2/2; 5/9] START leaf_size=20, n_neighbors=20…
[CV 2/2; 6/9] START leaf_size=20, n_neighbors=50…
[CV 1/2; 2/9] END .leaf_size=10, n_neighbors=20;, score=0.674 total time=   0.4s
[CV 1/2; 7/9] START leaf_size=30, n_neighbors=5…
[CV 1/2; 6/9] END .leaf_size=20, n_neighbors=50;, score=0.568 total time=   0.4s
[CV 2/2; 7/9] START leaf_size=30, n_neighbors=5…
[CV 1/2; 1/9] END ..leaf_size=10, n_neighbors=5;, score=0.753 total time=   0.5s
[CV 2/2; 1/9] END ..leaf_size=10, n_neighbors=5;, score=0.770 total time=   0.5s
[CV 1/2; 8/9] START leaf_size=30, n_neighbors=20…
[CV 2/2; 8/9] START leaf_size=30, n_neighbors=20…
```

```
[CV 2/2; 2/9] END .leaf_size=10, n_neighbors=20;, score=0.679 total time=   0.6s
[CV 1/2; 9/9] START leaf_size=30, n_neighbors=50…
[CV 2/2; 5/9] END .leaf_size=20, n_neighbors=20;, score=0.679 total time=   0.6s
[CV 1/2; 3/9] END .leaf_size=10, n_neighbors=50;, score=0.568 total time=   0.7s
[CV 2/2; 9/9] START leaf_size=30, n_neighbors=50…
[CV 2/2; 4/9] END ..leaf_size=20, n_neighbors=5;, score=0.770 total time=   0.7s
[CV 2/2; 3/9] END .leaf_size=10, n_neighbors=50;, score=0.592 total time=   0.7s
[CV 1/2; 4/9] END ..leaf_size=20, n_neighbors=5;, score=0.753 total time=   0.7s
[CV 1/2; 5/9] END .leaf_size=20, n_neighbors=20;, score=0.674 total time=   0.7s
[CV 2/2; 6/9] END .leaf_size=20, n_neighbors=50;, score=0.592 total time=   0.7s
[CV 1/2; 7/9] END ..leaf_size=30, n_neighbors=5;, score=0.753 total time=   0.5s
[CV 2/2; 7/9] END ..leaf_size=30, n_neighbors=5;, score=0.770 total time=   0.4s
[CV 2/2; 8/9] END .leaf_size=30, n_neighbors=20;, score=0.679 total time=   0.4s
[CV 1/2; 8/9] END .leaf_size=30, n_neighbors=20;, score=0.674 total time=   0.4s
[CV 2/2; 9/9] END .leaf_size=30, n_neighbors=50;, score=0.592 total time=   0.3s
[CV 1/2; 9/9] END .leaf_size=30, n_neighbors=50;, score=0.568 total time=   0.4s
```

```
 20%|         | 1/5 [00:01<00:06,  1.67s/it]
```

```
Best Score: 0.7612438498239373
Best Params: {'leaf_size': 10, 'n_neighbors': 5}
********************************************************************************
********************
Evaluating Model DecisionTreeClassifier
Fitting 2 folds for each of 3 candidates, totalling 6 fits
[CV 1/2; 2/3] START min_weight_fraction_leaf=0.2…
[CV 1/2; 1/3] START min_weight_fraction_leaf=0…
[CV 2/2; 2/3] START min_weight_fraction_leaf=0.2…
[CV 2/2; 1/3] START min_weight_fraction_leaf=0…
[CV 1/2; 3/3] START min_weight_fraction_leaf=0.5…
[CV 2/2; 3/3] START min_weight_fraction_leaf=0.5…
[CV 1/2; 3/3] END .min_weight_fraction_leaf=0.5;, score=0.003 total time=   0.1s
[CV 2/2; 3/3] END .min_weight_fraction_leaf=0.5;, score=0.001 total time=   0.1s
[CV 1/2; 2/3] END .min_weight_fraction_leaf=0.2;, score=0.007 total time=   0.2s
[CV 2/2; 2/3] END .min_weight_fraction_leaf=0.2;, score=0.013 total time=   0.2s
[CV 1/2; 1/3] END …min_weight_fraction_leaf=0;, score=0.402 total time=   0.8s
[CV 2/2; 1/3] END …min_weight_fraction_leaf=0;, score=0.416 total time=   0.9s
```

```
 40%|        | 2/5 [00:04<00:06,  2.18s/it]
```

```
Best Score: 0.4090657739714598
Best Params: {'min_weight_fraction_leaf': 0}
********************************************************************************
********************
Evaluating Model RandomForestClassifier
Fitting 2 folds for each of 3 candidates, totalling 6 fits
[CV 1/2; 1/3] START n_estimators=100…
[CV 2/2; 1/3] START n_estimators=100…
[CV 1/2; 2/3] START n_estimators=200…
[CV 2/2; 2/3] START
```

```
n_estimators=200…[CV 1/2; 3/3] START
n_estimators=300…

[CV 2/2; 3/3] START n_estimators=300…
[CV 2/2; 1/3] END …n_estimators=100;, score=0.812 total time=   2.0s
[CV 1/2; 1/3] END …n_estimators=100;, score=0.784 total time=   2.1s
[CV 1/2; 2/3] END …n_estimators=200;, score=0.798 total time=   3.9s
[CV 2/2; 2/3] END …n_estimators=200;, score=0.823 total time=   4.1s
[CV 1/2; 3/3] END …n_estimators=300;, score=0.805 total time=   5.8s
[CV 2/2; 3/3] END …n_estimators=300;, score=0.824 total time=   5.8s

 60%|       | 3/5 [00:21<00:17,  8.86s/it]

Best Score: 0.8144522073278911
Best Params: {'n_estimators': 300}
********************************************************************************
********************
Evaluating Model LGBMClassifier
Fitting 2 folds for each of 18 candidates, totalling 36 fits
[CV 1/2; 6/18] START learning_rate=0.1, max_depth=30, n_estimators=150…
[CV 2/2; 3/18] START learning_rate=0.1, max_depth=10, n_estimators=150…
[CV 1/2; 1/18] START learning_rate=0.1, max_depth=10, n_estimators=50…
[CV 2/2; 1/18] START learning_rate=0.1, max_depth=10, n_estimators=50…
[CV 2/2; 6/18] START learning_rate=0.1, max_depth=30, n_estimators=150…
[CV 1/2; 4/18] START learning_rate=0.1, max_depth=30, n_estimators=50…
[CV 1/2; 3/18] START learning_rate=0.1, max_depth=10, n_estimators=150…
[CV 2/2; 2/18] START learning_rate=0.1, max_depth=10, n_estimators=100…
[CV 2/2; 5/18] START learning_rate=0.1, max_depth=30, n_estimators=100…
[CV 2/2; 4/18] START learning_rate=0.1, max_depth=30, n_estimators=50…
[CV 1/2; 5/18] START learning_rate=0.1, max_depth=30,
n_estimators=100…[CV 1/2; 2/18] START learning_rate=0.1, max_depth=10,
n_estimators=100…

[CV 1/2; 4/18] END learning_rate=0.1, max_depth=30, n_estimators=50;,
score=0.694 total time= 2.9min
[CV 1/2; 7/18] START learning_rate=0.1, max_depth=-1, n_estimators=50…
[CV 2/2; 5/18] END learning_rate=0.1, max_depth=30, n_estimators=100;,
score=0.734 total time= 3.6min
[CV 2/2; 7/18] START learning_rate=0.1, max_depth=-1, n_estimators=50…
[CV 2/2; 1/18] END learning_rate=0.1, max_depth=10, n_estimators=50;,
score=0.713 total time= 3.7min
[CV 1/2; 8/18] START learning_rate=0.1, max_depth=-1, n_estimators=100…
[CV 2/2; 4/18] END learning_rate=0.1, max_depth=30, n_estimators=50;,
score=0.712 total time= 3.7min
[CV 2/2; 8/18] START learning_rate=0.1, max_depth=-1, n_estimators=100…
[CV 1/2; 1/18] END learning_rate=0.1, max_depth=10, n_estimators=50;,
score=0.691 total time= 3.7min
[CV 1/2; 9/18] START learning_rate=0.1, max_depth=-1, n_estimators=150…
[CV 2/2; 6/18] END learning_rate=0.1, max_depth=30, n_estimators=150;,
score=0.736 total time= 3.8min
[CV 2/2; 9/18] START learning_rate=0.1, max_depth=-1, n_estimators=150…
[CV 1/2; 3/18] END learning_rate=0.1, max_depth=10, n_estimators=150;,
score=0.728 total time= 4.0min
[CV 1/2; 10/18] START learning_rate=0.3, max_depth=10, n_estimators=50…
[CV 2/2; 3/18] END learning_rate=0.1, max_depth=10, n_estimators=150;,
score=0.738 total time= 4.0min
[CV 2/2; 10/18] START learning_rate=0.3, max_depth=10, n_estimators=50…
[CV 2/2; 2/18] END learning_rate=0.1, max_depth=10, n_estimators=100;,
score=0.736 total time= 4.2min
[CV 1/2; 11/18] START learning_rate=0.3, max_depth=10, n_estimators=100…
[CV 1/2; 5/18] END learning_rate=0.1, max_depth=30, n_estimators=100;,
score=0.717 total time= 4.3min
[CV 2/2; 11/18] START learning_rate=0.3, max_depth=10, n_estimators=100…
[CV 1/2; 2/18] END learning_rate=0.1, max_depth=10, n_estimators=100;,
score=0.726 total time= 4.4min
[CV 1/2; 12/18] START learning_rate=0.3, max_depth=10, n_estimators=150…
[CV 1/2; 10/18] END learning_rate=0.3, max_depth=10, n_estimators=50;,
score=0.041 total time=  31.0s
[CV 2/2; 12/18] START learning_rate=0.3, max_depth=10, n_estimators=150…
[CV 2/2; 10/18] END learning_rate=0.3, max_depth=10, n_estimators=50;,
score=0.058 total time=  32.7s
[CV 1/2; 13/18] START learning_rate=0.3, max_depth=30, n_estimators=50…
[CV 1/2; 6/18] END learning_rate=0.1, max_depth=30, n_estimators=150;,
score=0.719 total time= 4.7min
[CV 2/2; 13/18] START learning_rate=0.3, max_depth=30, n_estimators=50…
[CV 1/2; 11/18] END learning_rate=0.3, max_depth=10, n_estimators=100;,
score=0.041 total time=  40.1s
[CV 1/2; 14/18] START learning_rate=0.3, max_depth=30, n_estimators=100…
[CV 2/2; 13/18] END learning_rate=0.3, max_depth=30, n_estimators=50;,
score=0.068 total time=  23.3s
[CV 2/2; 14/18] START learning_rate=0.3, max_depth=30, n_estimators=100…
[CV 2/2; 11/18] END learning_rate=0.3, max_depth=10, n_estimators=100;,
score=0.058 total time=  44.1s
[CV 1/2; 15/18] START learning_rate=0.3, max_depth=30, n_estimators=150…
[CV 1/2; 13/18] END learning_rate=0.3, max_depth=30, n_estimators=50;,
score=0.068 total time=  31.4s
[CV 2/2; 15/18] START learning_rate=0.3, max_depth=30, n_estimators=150…
[CV 2/2; 12/18] END learning_rate=0.3, max_depth=10, n_estimators=150;,
score=0.058 total time=  40.9s
[CV 1/2; 16/18] START learning_rate=0.3, max_depth=-1, n_estimators=50…
[CV 1/2; 12/18] END learning_rate=0.3, max_depth=10, n_estimators=150;,
score=0.041 total time=  49.7s
[CV 2/2; 16/18] START learning_rate=0.3, max_depth=-1, n_estimators=50…
[CV 1/2; 14/18] END learning_rate=0.3, max_depth=30, n_estimators=100;,
score=0.068 total time=  42.9s
[CV 1/2; 17/18] START learning_rate=0.3, max_depth=-1, n_estimators=100…
[CV 2/2; 14/18] END learning_rate=0.3, max_depth=30, n_estimators=100;,
score=0.068 total time=  35.0s
[CV 2/2; 17/18] START learning_rate=0.3, max_depth=-1, n_estimators=100…
[CV 1/2; 7/18] END learning_rate=0.1, max_depth=-1, n_estimators=50;,
score=0.694 total time= 2.7min
[CV 1/2; 18/18] START learning_rate=0.3, max_depth=-1, n_estimators=150…
[CV 1/2; 15/18] END learning_rate=0.3, max_depth=30, n_estimators=150;,
score=0.068 total time=  39.9s
[CV 2/2; 18/18] START learning_rate=0.3, max_depth=-1, n_estimators=150…
[CV 1/2; 16/18] END learning_rate=0.3, max_depth=-1, n_estimators=50;,
score=0.068 total time=  31.7s
[CV 2/2; 15/18] END learning_rate=0.3, max_depth=30, n_estimators=150;,
score=0.068 total time=  40.7s
[CV 2/2; 16/18] END learning_rate=0.3, max_depth=-1, n_estimators=50;,
score=0.068 total time=  32.3s
[CV 2/2; 18/18] END learning_rate=0.3, max_depth=-1, n_estimators=150;,
score=0.068 total time=  27.5s
[CV 1/2; 17/18] END learning_rate=0.3, max_depth=-1, n_estimators=100;,
score=0.068 total time=  34.3s
[CV 2/2; 7/18] END learning_rate=0.1, max_depth=-1, n_estimators=50;,
score=0.712 total time= 2.6min
[CV 2/2; 17/18] END learning_rate=0.3, max_depth=-1, n_estimators=100;,
score=0.068 total time=  34.7s
[CV 1/2; 18/18] END learning_rate=0.3, max_depth=-1, n_estimators=150;,
score=0.068 total time=  37.3s
[CV 2/2; 8/18] END learning_rate=0.1, max_depth=-1, n_estimators=100;,
score=0.734 total time= 3.0min
[CV 1/2; 8/18] END learning_rate=0.1, max_depth=-1, n_estimators=100;,
score=0.717 total time= 3.1min
[CV 2/2; 9/18] END learning_rate=0.1, max_depth=-1, n_estimators=150;,
score=0.736 total time= 3.1min
[CV 1/2; 9/18] END learning_rate=0.1, max_depth=-1, n_estimators=150;,
score=0.719 total time= 3.2min
```

```
 80%|      | 4/5 [08:46<03:24, 204.83s/it]
```

```
Best Score: 0.7327590297920007
Best Params: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 150}
********************************************************************************
********************
Evaluating Model SVC
Fitting 2 folds for each of 6 candidates, totalling 12 fits
[CV 1/2; 1/6] START decision_function_shape=ovo, degree=3…
[CV 2/2; 1/6] START decision_function_shape=ovo, degree=3…
[CV 1/2; 2/6] START decision_function_shape=ovo, degree=10…
[CV 2/2; 2/6] START decision_function_shape=ovo, degree=10…
[CV 1/2; 3/6] START decision_function_shape=ovo, degree=20…
[CV 2/2; 3/6] START decision_function_shape=ovo, degree=20…
[CV 1/2; 4/6] START decision_function_shape=ovr, degree=3…
[CV 2/2; 4/6] START decision_function_shape=ovr, degree=3…
```

```
[CV 1/2; 5/6] START decision_function_shape=ovr, degree=10…
[CV 2/2; 5/6] START decision_function_shape=ovr, degree=10…
[CV 1/2; 6/6] START decision_function_shape=ovr, degree=20…
[CV 2/2; 6/6] START decision_function_shape=ovr, degree=20…
[CV 1/2; 6/6] END decision_function_shape=ovr, degree=20;, score=0.819 total
time=  12.6s
[CV 1/2; 3/6] END decision_function_shape=ovo, degree=20;, score=0.819 total
time=  13.2s
[CV 2/2; 4/6] END decision_function_shape=ovr, degree=3;, score=0.826 total
time=  13.7s
[CV 2/2; 5/6] END decision_function_shape=ovr, degree=10;, score=0.826 total
time=  14.3s
[CV 1/2; 1/6] END decision_function_shape=ovo, degree=3;, score=0.819 total
time=  15.4s
[CV 2/2; 6/6] END decision_function_shape=ovr, degree=20;, score=0.826 total
time=  15.6s
[CV 1/2; 4/6] END decision_function_shape=ovr, degree=3;, score=0.819 total
time=  15.6s
[CV 1/2; 5/6] END decision_function_shape=ovr, degree=10;, score=0.819 total
time=  15.7s
[CV 2/2; 2/6] END decision_function_shape=ovo, degree=10;, score=0.826 total
time=  15.7s
[CV 1/2; 2/6] END decision_function_shape=ovo, degree=10;, score=0.819 total
time=  15.8s
[CV 2/2; 3/6] END decision_function_shape=ovo, degree=20;, score=0.826 total
time=  15.9s
[CV 2/2; 1/6] END decision_function_shape=ovo, degree=3;, score=0.826 total
time=  16.0s
```

```
100%|     | 5/5 [09:06<00:00, 109.39s/it]
```

```
Best Score: 0.8223916590174787
Best Params: {'decision_function_shape': 'ovo', 'degree': 3}
```

1.20 Compare model scores

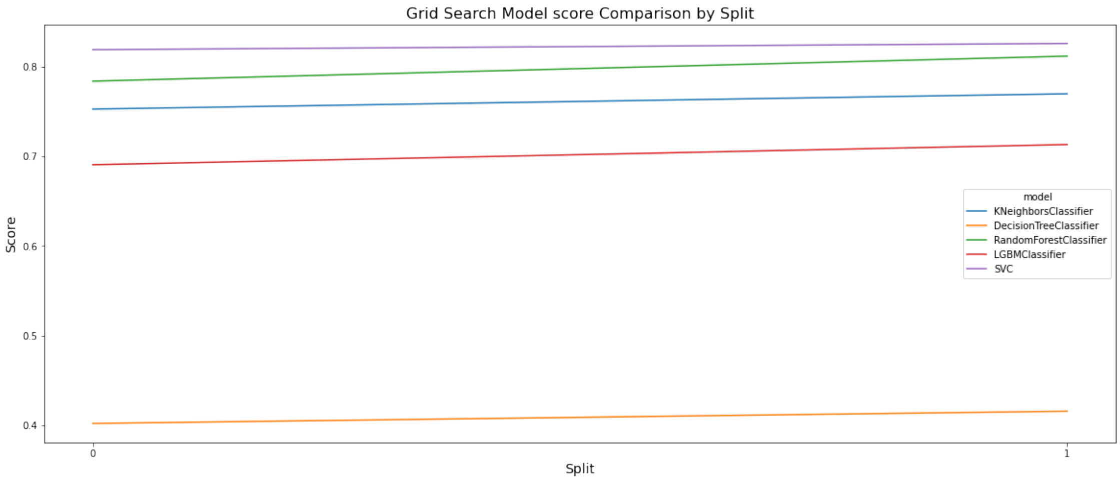
[ ]:

```
def get_split_scores(splits, grid_reg):
    scores = []
    models = []
    split_nums = []
    for i in range(splits):
        scores.append(grid_reg.cv_results_["split{}_test_score".format(i)][0])
        models.append(grid_reg.best_estimator_.__class__.__name__)
        split_nums.append(i)
    return split_nums, scores, models

splits = cv.cvargs["n_splits"]
```

```
fig, ax =plt.subplots(1,figsize=(20, 8))
plt.xticks(range(splits))
plt.xlabel("Split", fontsize=14)
plt.ylabel("Score", fontsize=14)
plt.title("Grid Search Model score Comparison by Split", fontsize=16)
scores = []
models = []
split_nums = []
for reg in grid_result:
    t_split, t_scores, t_models = get_split_scores(splits, reg)
    scores = np.append(scores, t_scores)
    models = np.append(models, t_models)
    split_nums = np.append(split_nums, t_split)
result = pd.DataFrame()
result["split"] = split_nums
result["model"] = models
result["score"] = scores
g = sns.lineplot(x=result.split, y=result.score, hue=result.model)
plt.show
```

```
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>
```

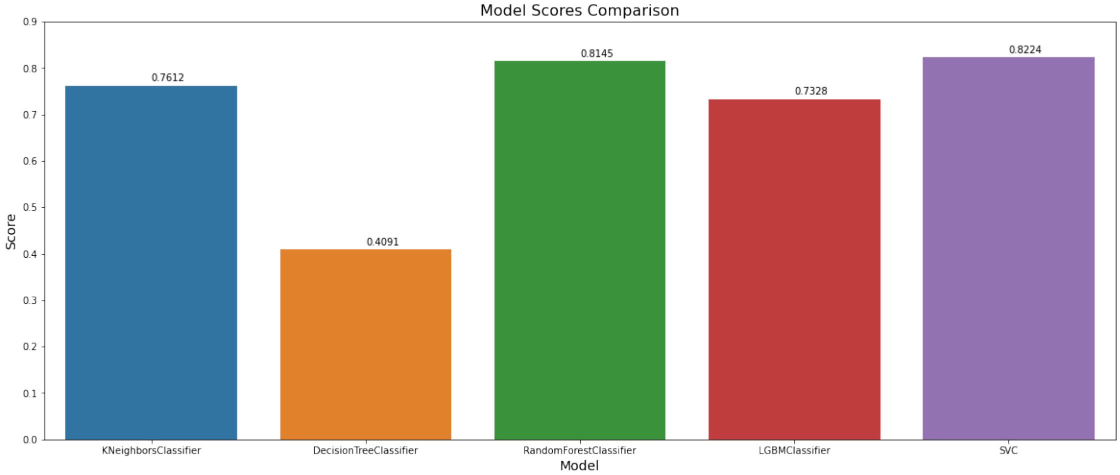


1.21 Best Scores

```
[ ]: best_scores = [x.best_score_ for x in grid_result]
model_names = [x.best_estimator_.__class__.__name__ for x in grid_result]
fig, ax =plt.subplots(1,figsize=(20, 8))
g=sns.barplot(model_names, best_scores)
plt.title("Model Scores Comparison", fontsize=16)
plt.xlabel("Model", fontsize=14)
plt.ylabel("Score", fontsize=14)
plt.yticks(np.arange(0.0, 1.0, 0.1))

for i in range(len(model_names)):
    g.text(i, best_scores[i]+0.01, "{:0.4f}".format(best_scores[i]))

plt.show()
```



1.22 Train final model

```
[ ]: model = svm.SVC(random_state= random_state, decision_function_shape= 'ovo',␣
      ↪degree= 3)
     result = model.fit(train_X, train_Y, )
```

1.23 Test model with Test Set

```
[ ]: #pred = model.predict(test_X)
     from sklearn.model_selection import cross_val_score
     cross_val_score(model, test_X, test_Y, cv=2, scoring="f1_macro")
```

```
[ ]: array([0.62942225, 0.66207964])
```

```
[ ]: prediction = model.predict(test_X)
```

```
[ ]: pred_labels = labelEncoder.inverse_transform(prediction)
     pred_labels
```

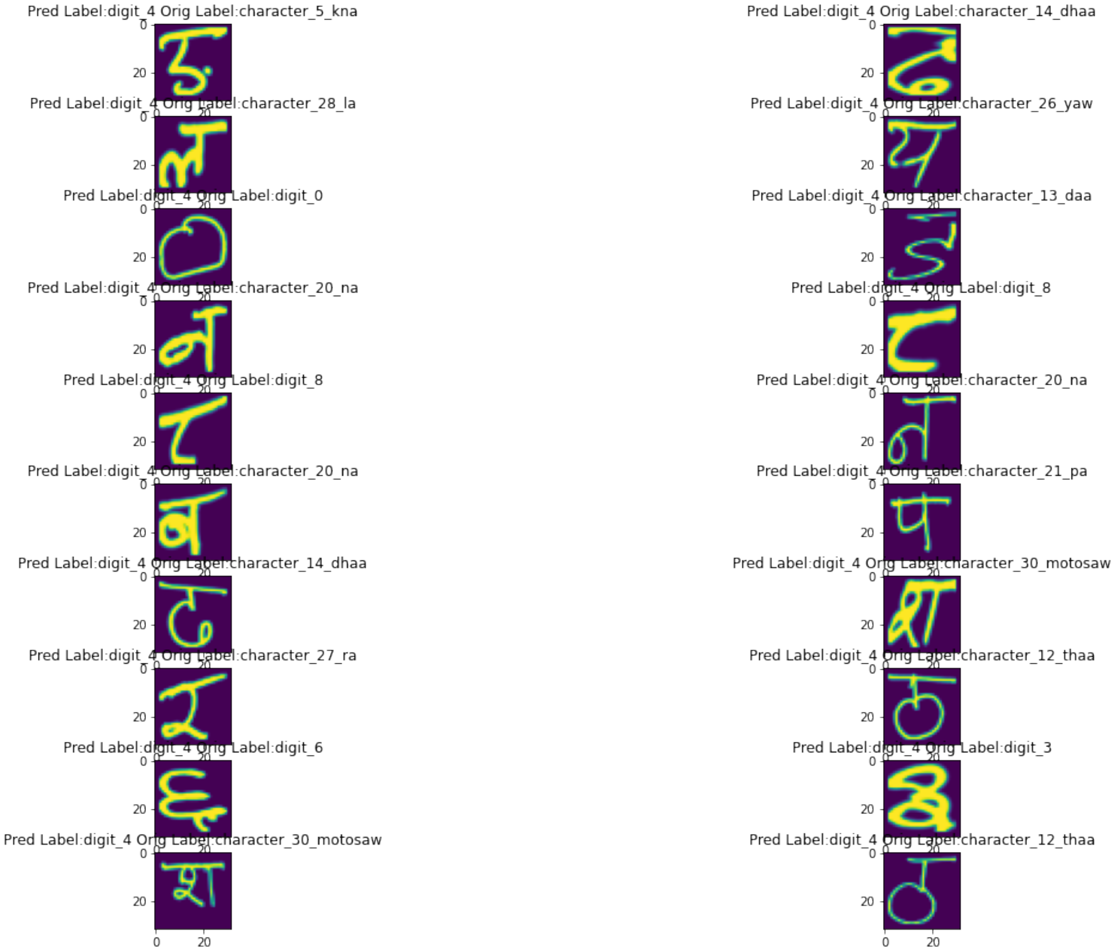
```
[ ]: array(['digit_4', 'digit_4', 'digit_4', …, 'digit_4', 'digit_4',
            'digit_4'], dtype=object)
```

```
[ ]: orig_labels = labelEncoder.inverse_transform(test_Y)
```

[ ]:
```
import random
rand = [random.randrange(0,len(test_Y)) for i in range(20)]
```

1.24 Sample Predictions

[ ]:
```
plt.figure(figsize=(20, 45))
for i in range(20):
    plt.subplot(32, 2, i+1)
    a = np.array(test_X.iloc[rand[i]][:1024], dtype='float')
    a = a.reshape(32,32)
    plt.title("Pred Label:{} Orig Label:{}".format(pred_labels[rand[i]],␣
 ↪orig_labels[rand[i]]))
    plt.imshow(a)
plt.show()
```


Pred Label:digit_4 Orig Label:character_5_kna
Pred Label:digit_4 Orig Label:character_14_dhaa
Pred Label:digit_4 Orig Label:character_28_la
Pred Label:digit_4 Orig Label:character_26_yaw
Pred Label:digit_4 Orig Label:digit_0
Pred Label:digit_4 Orig Label:character_13_daa
Pred Label:digit_4 Orig Label:character_20_na
Pred Label:digit_4 Orig Label:digit_8
Pred Label:digit_4 Orig Label:digit_8
Pred Label:digit_4 Orig Label:character_20_na
Pred Label:digit_4 Orig Label:character_20_na
Pred Label:digit_4 Orig Label:character_21_pa
Pred Label:digit_4 Orig Label:character_14_dhaa
Pred Label:digit_4 Orig Label:character_30_motosaw
Pred Label:digit_4 Orig Label:character_27_ra
Pred Label:digit_4 Orig Label:character_12_thaa
Pred Label:digit_4 Orig Label:digit_6
Pred Label:digit_4 Orig Label:digit_3
Pred Label:digit_4 Orig Label:character_30_motosaw
Pred Label:digit_4 Orig Label:character_12_thaa