

DSC680-devnagari-handwritten-chars-classification-scikit

February 21, 2025

1 Handwritten Devnagari Character classification

1.0.1 Author: Sheetal Munjewar

1.0.2 Bellevue University

1.1 Import Libraries

```
[1]: 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

```
[2]: random_state = 17
np.random.seed(random_state)
import warnings
warnings.filterwarnings('ignore')
```

1.3 Define datasource paths

```
[3]: 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.

```
[4]: 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

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

100%|

| 46/46 [00:00<00:00, 104.38it/s]

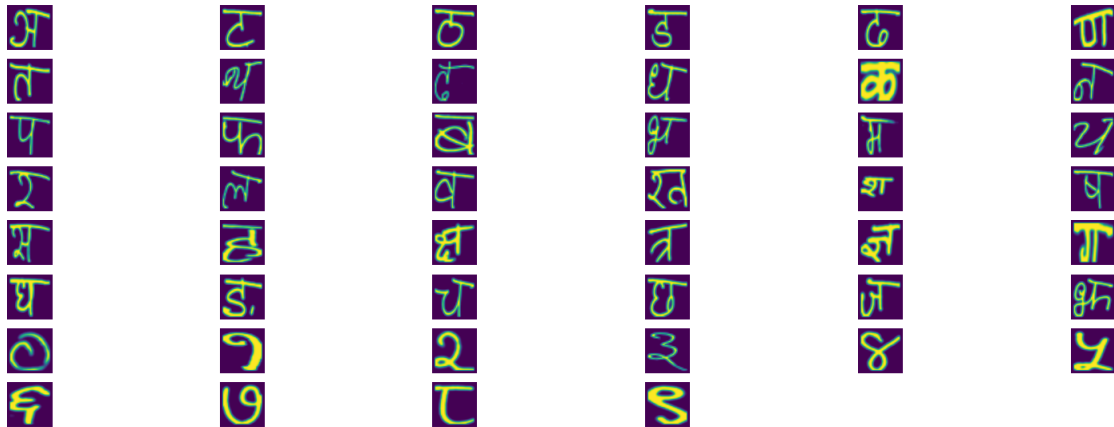
```
[5]: (46, 32, 32)
```

1.6 Function to Display Images

```
[6]: 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

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



1.8 Load Training and Test Dataset

```
[8]: 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 [00:37<00:00, 1.23it/s]

Loading test data...

100%|

| 46/46 [00:07<00:00, 6.40it/s]

1.9 Display Dataset shapes

```
[9]: 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

```
[10]: 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])
```

```
[11]: 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

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

```
[13]: train_df
```

```
[13]:
```

	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_10_yna
1	0	0	0	character_10_yna
2	0	0	0	character_10_yna
3	0	0	0	character_10_yna
4	0	0	0	character_10_yna
...
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]

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

```
[15]: test_df
```

```
[15]:
```

	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_10_yna
1	0	0	0	character_10_yna
2	0	0	0	character_10_yna
3	0	0	0	character_10_yna
4	0	0	0	character_10_yna
...
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

```
[16]: 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
```

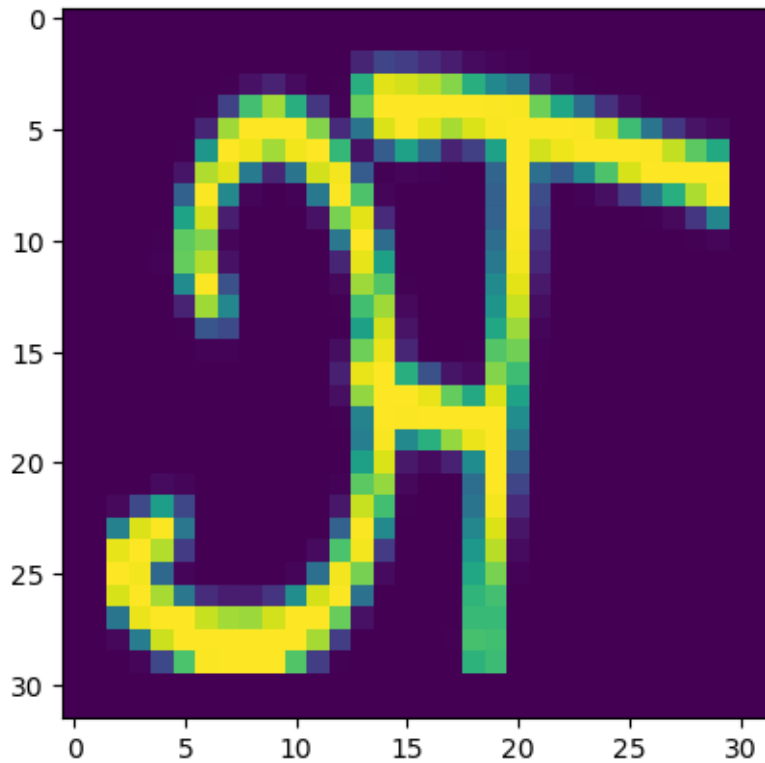
```
[17]: normalizer = Normalizer("label")  
train_new = normalizer.transform(train_df)  
#train_new[122]
```

100%|

| 1025/1025 [00:13<00:00, 73.38it/s]

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

```
[18]: <matplotlib.image.AxesImage at 0x19c2b0a6830>
```



1.13 Class for label encoding

```
[19]: 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

```
[20]: labelEncoder = MyLabelEncoder("label")
labelEncoder.fit(train_df)

labelEncoder.classes_
# array([1, 2, 6])
```

```
#labelEncoder.transform(['character_10_yna', 'character_11_taaamatar',
↪ 'character_12_thaa', 'character_13_daa'])
# array([0, 0, 1, 2]...)
# >>> le.inverse_transform([0, 0, 1, 2])
# array([1, 1, 2, 6])
```

```
[20]: array(['character_10_yna', 'character_11_taaamatar', '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

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

1.16 Apply transformation pipeline

```
[22]: train_df = transform_pipeline.transform(train_df)

test_df = transform_pipeline.transform(test_df)
```

100%|

| 1025/1025 [00:00<00:00, 3315.74it/s]

100%|

| 1025/1025 [00:03<00:00, 291.22it/s]

1.17 Split input and labels in separate dataframes

```
[23]: 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"]
```



```
[24]: cv = RepeatedStratifiedKFold(n_splits=2, n_repeats=1, random_state=random_state)
```

1.18 Create sample dataset for grid search

```
[25]: #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

```
[26]: 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

20%|
| 1/5 [00:05<00:20, 5.14s/it]

Best Score: nan
Best Params: {'leaf_size': 10, 'n_neighbors': 5}
*****
*****
Evaluating Model DecisionTreeClassifier
Fitting 2 folds for each of 3 candidates, totalling 6 fits

40%|
| 2/5 [00:09<00:13, 4.64s/it]

Best Score: 0.4922791856275067
Best Params: {'min_weight_fraction_leaf': 0}
*****
*****
Evaluating Model RandomForestClassifier
Fitting 2 folds for each of 3 candidates, totalling 6 fits

60%|

| 3/5 [00:40<00:33, 16.64s/it]

Best Score: 0.8799995937635741
Best Params: {'n_estimators': 300}
*****
*****
Evaluating Model LGBMClassifier

```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

80%|

| 4/5 [17:42<06:53, 413.74s/it]

Best Score: 0.8071037210650509

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

100%|

| 5/5 [18:12<00:00, 218.46s/it]

Best Score: 0.8916885009708901

Best Params: {'decision_function_shape': 'ovo', 'degree': 3}

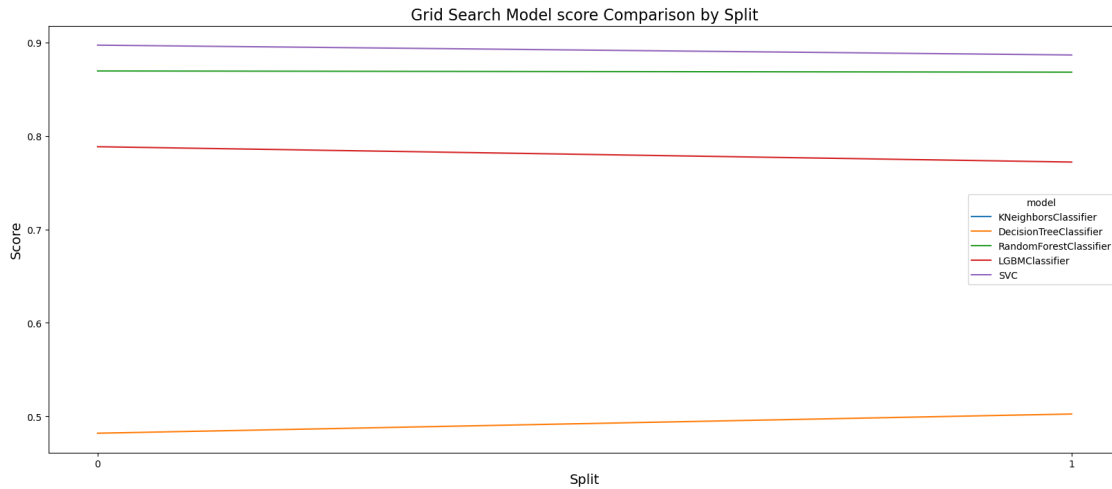
1.20 Compare model scores

```
[27]: 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
```

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



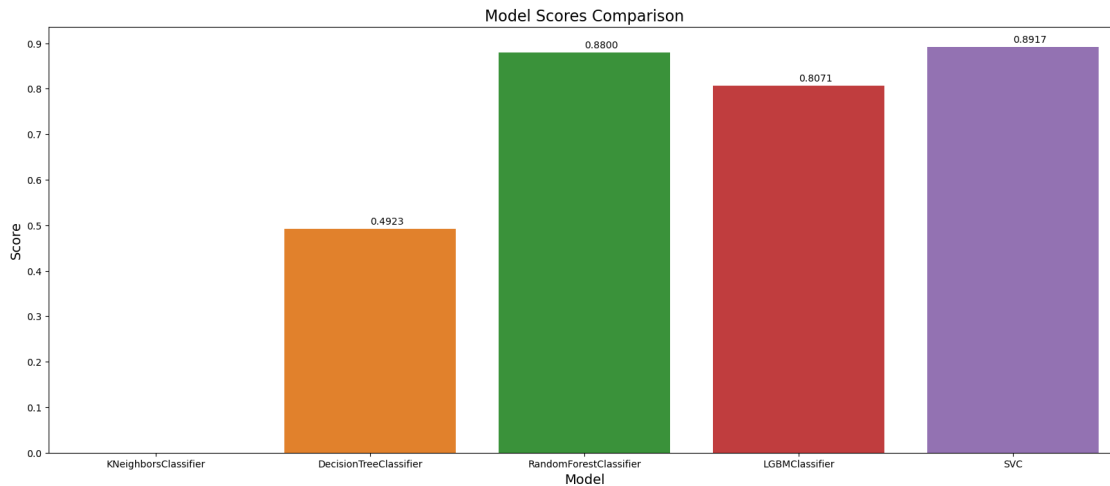
1.21 Best Scores

```
[37]: 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(x=model_names, y=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()
```

posx and posy should be finite values
 posx and posy should be finite values



1.22 Train final model

```
[30]: 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

```
[31]: #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")
```

```
[31]: array([0.60690618, 0.60475535])
```

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

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

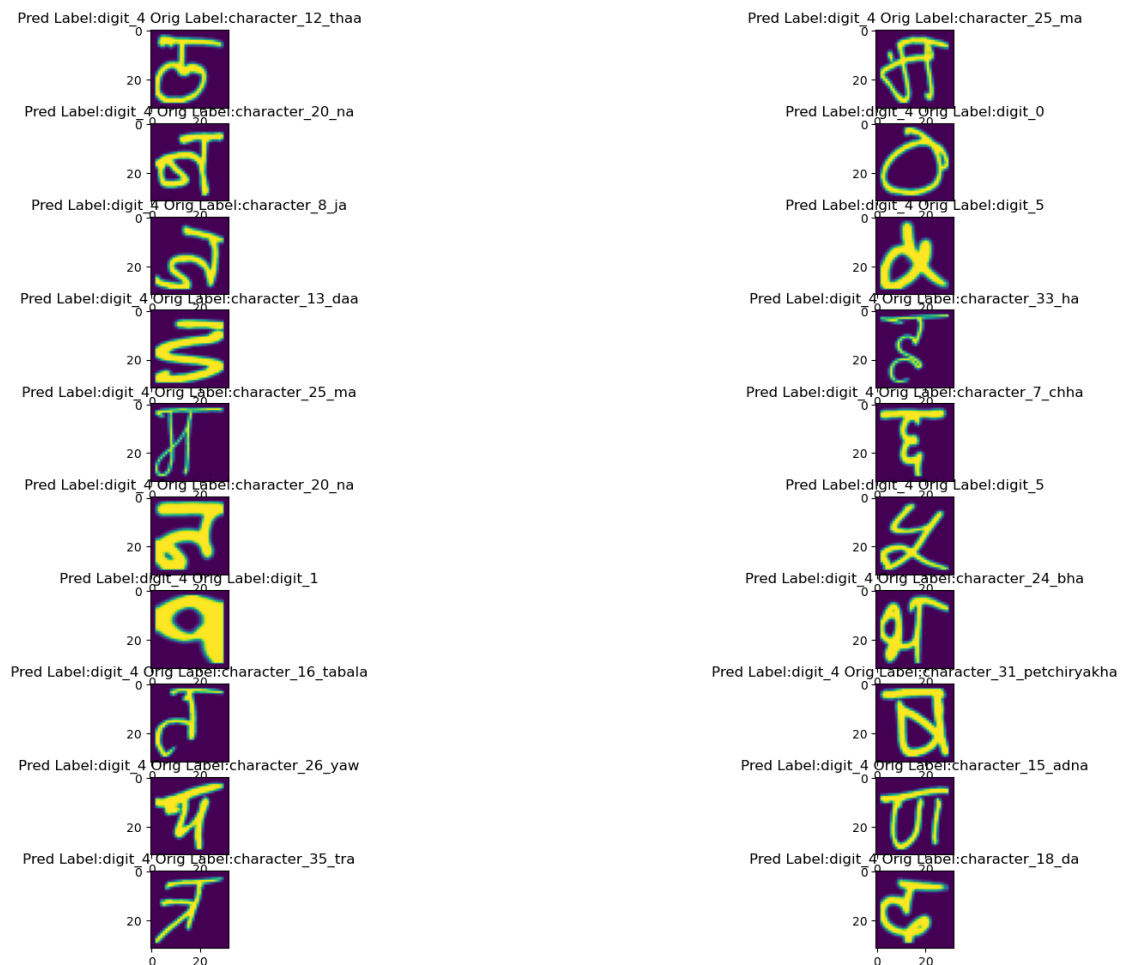
```
[33]: array(['digit_4', 'digit_4', 'digit_4', ..., 'digit_4', 'digit_4',
    'digit_4'], dtype=object)
```

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

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

1.24 Sample Predictions

```
[36]: 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()
```



[]: