Machine Learning Project Documentation

Sudanese Primary Schools Dataset Analysis and Classification based on Facility Availability

Model Refinement

1. Overview

The Model Refinement phase plays a pivotal role in enhancing the performance of the machine learning model, which aims to classify Sudanese primary schools based on facility availability. This phase is critical for ensuring the model's accuracy and relevance to address educational challenges in Sudan, aligning with Sustainable Development Goals 4, 5, and 9.

2. Model Evaluation

In the initial model evaluation, results have been observed. however, areas for improvement were identified. Key metrics, such as precision, recall, and F1-score, along with visualizations from the neural network model steps, guided the focus on refining specific aspects of the model.

3. Refinement Techniques and Hyperparameter Tuning

To refine the model, various techniques have been employed.

- For Naïve bayes model, fitting 10 folds for each of 25 candidates, totaling 250 fits have been done after running the model for different fold number to check the highest accuracy.
- For random forest model it found that 50 number of estimators give the best performance.
- For KNN model, the most important parameter in this model is the number of k neighbors. A small code has been done to check which k value gives the best accuracy.
- For SVM model, kernel type changed until found that linear kernel gives the highest accuracy.
- For NN model, epochs 10 with 32 batch size gives the best accuracy.

6. Feature Selection

For feature selection, from sklearn.feature_selection import RFE library have been used to find the most effective features. For given features it rank them as fellow, Features: 'School ID', 'STCODE', 'LOCCODE', 'location', 'Type', 'Status', 'teachers', 'students_f', 'students_m', 'students_total', 'Total_Classrooms', 'Permanent', 'Needs Rehabilitation', 'Fence', 'store', 'school_feeding', 'kindergarten', 'kinder_level1', 'kinder_level2', 'electricity', 'Potable_Water_source', 'Latrines', 'Latrine_male', 'Latrine_female', 'Latrine_common', 'PCA1','PCA2'

```
Feature: 0, Selected False, Rank: 4.0
Feature: 1, Selected False, Rank: 19.0
Feature: 2, Selected False, Rank: 11.0
Feature: 3, Selected False, Rank: 21.0
Feature: 4, Selected False, Rank: 5.0
Feature: 5, Selected True, Rank: 1.0
Feature: 6, Selected False, Rank: 8.0
Feature: 7, Selected False, Rank: 10.0
Feature: 8, Selected False, Rank: 17.0
Feature: 9, Selected False, Rank: 2.0
Feature: 10, Selected True, Rank: 1.0
Feature: 11, Selected False, Rank: 3.0
Feature: 12, Selected False, Rank: 14.0
Feature: 13, Selected False, Rank: 6.0
Feature: 14, Selected False, Rank: 15.0
Feature: 15, Selected False, Rank: 9.0
Feature: 16, Selected False, Rank: 22.0
Feature: 17, Selected False, Rank: 13.0
Feature: 18, Selected False, Rank: 16.0
Feature: 19, Selected True, Rank: 1.0
Feature: 20, Selected False, Rank: 7.0
Feature: 21, Selected True, Rank: 1.0
Feature: 22, Selected False, Rank: 18.0
Feature: 23, Selected False, Rank: 12.0
Feature: 24, Selected False, Rank: 20.0
Feature: 25, Selected True, Rank: 1.0
Feature: 26, Selected True, Rank: 1.0
```

Test Submission

1. Overview

The Test Submission phase involves preparing the model for deployment or evaluation on a test dataset, marking a crucial step toward addressing educational challenges in Sudan.

2. Data Preparation for Testing

The test dataset was accurately prepared to ensure alignment with real-world scenarios. Considerations were made to account for potential variations and challenges that may be encountered in practice. The same data preparation techniques for training data done for testing as well; missing data check and converting categorical to numerical.

As the dataset is quit big, drop all missing data rows have been done
From 19379 ==> 18716 only 663 rows have been extracted

In [26]:
```
df = mydataset.dropna()
df.shape
```

Out[26]:
```
(18716, 38)
```

1st and 5th columns written in Arabic and as both have similer information column written in english ==> Drop

In [27]:
```
col_delete = [1,5]
col_delete = df.columns[col_delete]

# Drop
df = df.drop(columns=col_delete, errors='ignore')
```

School name, state name, location (LOCENG) ===> drop

In [28]:
```
col_delete = [1,2,4]
col_delete = df.columns[col_delete]

# Drop
df = df.drop(columns=col_delete, errors='ignore')
```

School grade1 to grade8 columns ==> drop

In [29]:
```
col_delete = [10,11,12,13,14,15,16,17]
col_delete = df.columns[col_delete]

# Drop
df = df.drop(columns=col_delete, errors='ignore')
```

edit STCODE and LOCCODE columns to be only the numerical part without "SD"

In [31]:
```python
df['STCODE'] = df['STCODE'].str.extract('(\d+)')
df['LOCCODE'] = df['LOCCODE'].str.extract('(\d+)')
# Convert the result to numeric
df['STCODE'] = pd.to_numeric(df['STCODE'], errors='coerce')
df['LOCCODE'] = pd.to_numeric(df['LOCCODE'], errors='coerce')
df
```

Out[31]:

	School ID	STCODE	LOCCODE	location	Type	Status	teachers	students_f	students_m	students_tc
0	53411301	18	18104	urban	Boys	normal	14	0	486	4
1	53411302	18	18104	urban	Boys	normal	14	0	360	3
2	53411303	18	18104	rural	Boys	normal	7	0	454	4
3	53411304	18	18104	rural	Boys	normal	7	0	445	4
4	53411305	18	18104	urban	Girls	normal	13	443	0	4

Convert yes,no to 1,0 in columns; Fence, Store, School_feeding, Latrines

In [32]:
```python
columns_to_replace = ['Fence', 'store', 'school_feeding','kindergarten', 'Latrines']

# Replace 'yes' with 1 and 'no' with 0
df[columns_to_replace] = df[columns_to_replace].replace({'yes': 1, 'no': 0})
df
```

Out[32]:

	School ID	STCODE	LOCCODE	location	Type	Status	teachers	students_f	students_m	students_total	...	school_feeding	kindergarten	k
0	53411301	18	18104	urban	Boys	normal	14	0	486	486	...	0	1	
1	53411302	18	18104	urban	Boys	normal	14	0	360	360	...	1	1	
2	53411303	18	18104	rural	Boys	normal	7	0	454	454	...	0	1	
3	53411304	18	18104	rural	Boys	normal	7	0	445	445	...	0	1	

check unique classes in 'location', 'Type', 'Status' , 'electricity', 'Potable_Water_source' columns convert them to numerical

In [33]:
```python
columns_of_interest = ['location', 'Type', 'Status' , 'electricity', 'Potable_Water_source']

for column in columns_of_interest:
    unique_classes = df[column].unique()
    print(f"Unique classes in {column}: {unique_classes}")
```

```
Unique classes in location: ['urban' 'rural']
Unique classes in Type: ['Boys' 'Girls' 'Mixed']
Unique classes in Status: ['normal' 'nomadic' 'nongovernmental' 'special needs' 'quranic'
 'complementary' 'displaced']
Unique classes in electricity: ['No' 'Solar energy' 'Generator' 'Public network']
Unique classes in Potable_Water_source: ['Well' 'Other' 'Pump' 'No' 'Public network']
```

In [34]:
```python
location_mapping = {'urban': 0, 'rural': 1}
df['location'] = df['location'].map(location_mapping)

type_mapping = {'Boys': 0, 'Girls': 1, 'Mixed': 2}
df['Type'] = df['Type'].map(type_mapping)

status_mapping = {'normal': 0, 'nomadic': 1, 'nongovernmental': 2, 'special needs': 3, 'quranic': 4,
                  'complementary': 5, 'displaced': 6}
df['Status'] = df['Status'].map(status_mapping)

electricity_mapping = {'No': 0, 'Solar energy': 1, 'Generator': 2, 'Public network': 3}
df['electricity'] = df['electricity'].map(electricity_mapping)

water_source_mapping = {'No': 0,'Well': 1,'Pump': 2,  'Public network': 3, 'Other': 4 }
df['Potable_Water_source'] = df['Potable_Water_source'].map(water_source_mapping)

df
```

3. Model Application

The trained model was applied to the test dataset using the best practices established during the refinement phase. The application process was streamlined to ensure efficiency and accuracy.

- Naïve bayes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV

params_NB = {'var_smoothing': np.logspace(0,-9, num=25)}
gnb = GridSearchCV(estimator=GaussianNB(), param_grid=params_NB, verbose=1, cv=10, n_jobs=-1)

#Train the model using the training sets
modelnb = gnb.fit(x_train, y_train)

target_predictednb=modelnb.predict(x_test)
print('\nClassification report\n', classification_report(y_test,target_predictednb))

matrix = confusion_matrix(y_test, target_predictednb)
print("Class Confusion Matrix\n", matrix)
```

- Random forest

```
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(n_estimators=50, random_state=0)
model_rf = random_forest.fit(x_train, y_train)
target_predicted_rf = model_rf.predict(x_test)

print('\nClassification report for Random Forest\n', classification_report(y_test, target_predicted_rf))
matrix_rf = confusion_matrix(y_test, target_predicted_rf)
print("Confusion Matrix for Random Forest:\n", matrix_rf)

```

- KNN

```
from sklearn.neighbors import KNeighborsClassifier

#print('#k','\t','Accuracy')
#for i in range (1,500):
 #       neigh = KNeighborsClassifier(n_neighbors=i)
  #      modelknn = neigh.fit(xtrain, ytrain)
   #     target_predictedknn=modelknn.predict(xval)
    #    print(i,'\t','%.3f' %(modelknn.score(xval, yval)*100),'%')


neigh = KNeighborsClassifier(n_neighbors=6)
modelknn = neigh.fit(x_train, y_train)
target_predictedknn=modelknn.predict(x_test)
print('\nClassification report with k=6\n', classification_report(y_test,target_predictedknn))
matrix = confusion_matrix(y_test, target_predictedknn)
print("Class Confusion Matrix\n", matrix)
```

- SVM

```
from sklearn.svm import SVC
clf = SVC(kernel='linear',gamma='auto')
modelsvm= clf.fit(x_train, y_train)
target_predictedsvm=modelsvm.predict(x_test)
print('\nClassification report with linear kernel\n', classification_report(y_test,target_predictedsvm))

matrix = confusion_matrix(y_test, target_predictedsvm)
print("Class Confusion Matrix\n", matrix)


```

- Neural network

```
num_classes = y_train.shape[1]
y_train_one_hot = to_categorical(np.argmax(y_train, axis=1), num_classes=num_classes)
y_val_one_hot = to_categorical(np.argmax(y_val, axis=1), num_classes=num_classes)
y_test_one_hot = to_categorical(np.argmax(y_test, axis=1), num_classes=num_classes)

# Build model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(x_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train
model.fit(x_train, y_train_one_hot, epochs=10, batch_size=32, validation_data=(x_val, y_val_one_hot))

# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test_one_hot)
print(f'Test accuracy: {test_acc}')
```

- Linear regression

```
# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model.predict(x_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print coefficients and evaluation metrics
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

4. Test Metrics

Evaluation metrics, including precision, recall, and F1-score, were employed to assess the model's performance on the test dataset. For linear regression only MSE metric have been done.

Table below contains comparative details about test data evaluation.

Training set = (14972, 6) (14972,)
Test set = (3744, 6) (3744,)

Model	Metrices
Naïve bayes	Fitting 10 folds for each of 25 candidates, totalling 250 fits Classification report precision recall f1-score support 0 0.90 0.89 0.89 1154 1 0.96 0.96 0.96 929 2 0.89 0.93 0.91 912 3 0.86 0.74 0.80 172 4 0.96 0.97 0.96 577 accuracy 0.92 3744 macro avg 0.91 0.90 0.90 3744 weighted avg 0.92 0.92 0.92 3744 Class Confusion Matrix [[1022 31 93 3 5] [35 892 0 2 0] [45 2 846 9 10] [30 1 5 128 8] [1 5 6 7 558]]
Random forest	Classification report for Random Forest precision recall f1-score support 0 0.94 0.95 0.94 1154 1 0.98 0.98 0.98 929 2 0.96 0.94 0.95 912 3 0.92 0.90 0.91 172 4 0.99 0.99 0.99 577 accuracy 0.96 3744 macro avg 0.96 0.95 0.95 3744 weighted avg 0.96 0.96 0.96 3744 Confusion Matrix for Random Forest: [[1091 13 39 11 0] [14 915 0 0 0] [43 2 861 0 6] [15 0 1 155 1] [3 2 0 3 569]]
KNN	Classification report with k=6 precision recall f1-score support 0 0.92 0.95 0.93 1154 1 0.98 0.97 0.98 929 2 0.95 0.93 0.94 912 3 0.95 0.89 0.92 172 4 0.99 0.98 0.99 577 accuracy 0.95 3744 macro avg 0.96 0.95 0.95 3744 weighted avg 0.95 0.95 0.95 3744 Class Confusion Matrix [[1097 15 36 6 0] [22 905 1 0 1] [60 2 848 0 2] [17 0 0 153 2] [2 2 4 2 567]]

<table>
<tr>
<td>SVM</td>
<td><pre>
Classification report with linear kernel
              precision   recall f1-score   support

           0   0.94   0.94   0.94   1154
           1   0.98   0.99   0.98   929
           2   0.95   0.95   0.95   912
           3   0.93   0.93   0.93   172
           4   0.99   0.98   0.99   577

    accuracy   0.96   3744
   macro avg   0.96   0.96   0.96   3744
weighted avg   0.96   0.96   0.96   3744

Class Confusion Matrix
 [[1087   14   44   9   0]
 [11 918   0   0   0]
 [42   2 865   0   3]
 [10   0   0 160   2]
 [4   2   2   3 566]]
</pre></td>
</tr>
<tr>
<td>Neural network
Training set = (11977, 27) (11977, 5)
Validation set = (2995, 27)

(2995, 5)
Test set = (3744, 27) (3744, 5)</td>
<td><pre>
Epoch 1/10
375/375 [==============================] - 2s 3ms/step - loss: 0.3425 - accuracy: 0.9026 - val_loss: 0.0992 - val_accuracy: 0.9
659
Epoch 2/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0686 - accuracy: 0.9765 - val_loss: 0.0599 - val_accuracy: 0.9
773
Epoch 3/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0467 - accuracy: 0.9836 - val_loss: 0.0568 - val_accuracy: 0.9
796
Epoch 4/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0389 - accuracy: 0.9869 - val_loss: 0.0429 - val_accuracy: 0.9
816
Epoch 5/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0313 - accuracy: 0.9891 - val_loss: 0.0390 - val_accuracy: 0.9
806
Epoch 6/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0285 - accuracy: 0.9896 - val_loss: 0.0387 - val_accuracy: 0.9
820
Epoch 7/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0253 - accuracy: 0.9926 - val_loss: 0.0691 - val_accuracy: 0.9
820
Epoch 8/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0237 - accuracy: 0.9911 - val_loss: 0.0474 - val_accuracy: 0.9
843
Epoch 9/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0200 - accuracy: 0.9929 - val_loss: 0.0506 - val_accuracy: 0.9
860
Epoch 10/10
375/375 [==============================] - 1s 2ms/step - loss: 0.0185 - accuracy: 0.9938 - val_loss: 0.0308 - val_accuracy: 0.9
856
117/117 [==============================] - 0s 1ms/step - loss: 0.0490 - accuracy: 0.9848
Test accuracy: 0.9847756624221802
</pre>
<pre>
117/117 [==============================] - 0s 1ms/step
Classification Report:
              precision   recall f1-score   support

           0   0.99   0.97   0.98   1154
           1   0.99   1.00   0.99   929
           2   0.98   0.99   0.98   912
           3   0.94   0.98   0.96   172
           4   0.99   0.99   0.99   577

    accuracy   0.98   3744
   macro avg   0.98   0.98   0.98   3744
weighted avg   0.98   0.98   0.98   3744

Confusion Matrix:
[[1121   4   19   9   1]
 [4 925   0   0   0]
 [4   3 901   0   4]
 [3   0   1 168   0]
 [3   0   0   2 572]]
</pre></td>
</tr>
<tr>
<td>Linear regression</td>
<td><pre>Mean Squared Error: 0.04532975701283329</pre></td>
</tr>
</table>

5. Model Deployment

Regarding deploying the model in the real-world I am planning to **use Flask or Django** frameworks. The plan is to create a user interface where users can input school information, and the system will provide the corresponding classification group. Additionally, suggestions for facility improvement will be presented based on the model's insights. Integration with other systems or platforms was explored for future implementation.

6. Code Implementation

Clustering step done for making label schools depends on facility availability, elbow method done for choosing the best k value.

In [38]:

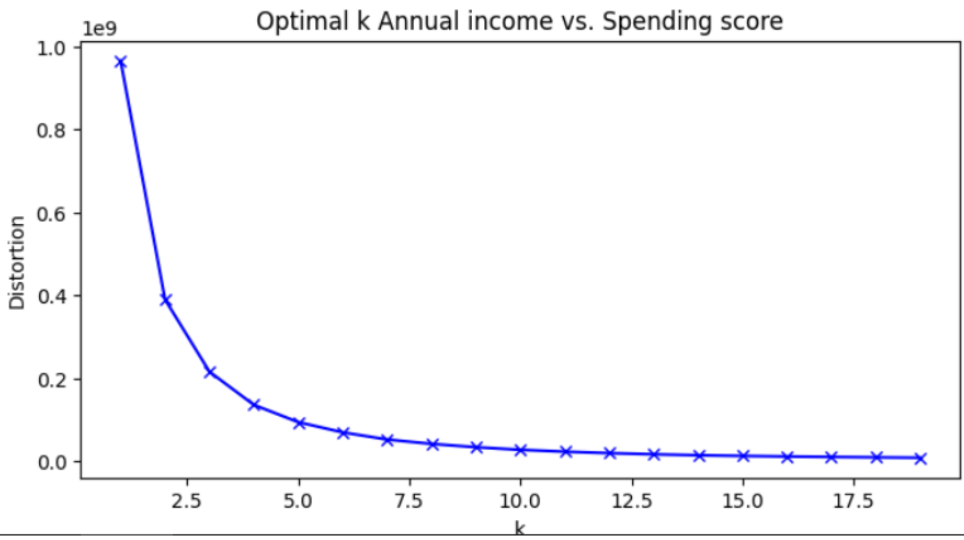
```
from sklearn.cluster import KMeans

distortions1 = []
K1= range(1,20)
for k1 in K1:
    kmeanModel = KMeans(n_clusters=k1, init='k-means++', random_state=0)
    kmeanModel.fit(x1)
    distortions1.append(kmeanModel.inertia_)

```

In [39]:

```
plt.figure(figsize=(8,4))
plt.plot(K1, distortions1, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('Optimal k Annual income vs. Spending score')
plt.show()
```



Then k-means applied to the data

```python
# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(x1)

n_clusters = 5

# Apply K-Means clustering
kmeans = KMeans(n_clusters= n_clusters, random_state=42)
clusters = kmeans.fit_predict(scaled_data)

# Add cluster labels to DataFrame
df['Cluster'] = clusters

# Visualize the clusters in 2D using PCA
# because features more than 2 PCA is done to find common between them and to ease visualizing
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_data)
df['PCA1'] = pca_result[:, 0]
df['PCA2'] = pca_result[:, 1]

plt.figure(figsize=(10, 6))
plt.scatter(df['PCA1'], df['PCA2'], c=df['Cluster'], cmap='viridis', edgecolor='k', s=50)
plt.title(f'K-Means Clustering with {n_clusters} Clusters')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.colorbar(label='Cluster')
plt.show()
```

Conclusion

In conclusion, the Model Refinement phase significantly improved the model's accuracy and robustness, laying the groundwork for addressing educational challenges in Sudan. The Test Submission phase provided valuable insights into the model's real-world applicability, with a focus on making user-friendly deployment through Flask or Django frameworks.