Heaven's Light is Our Guide

Computer Science and Engineering
Rajshahi University of Engineering and Technology

Course No: CSE 4204

Course Title: Sessional Based on CSE 4203

No of Experiment:01

Name of the Experiment: Implementation of Nearest Neighbor classification algorithms with and without distorted pattern.

Course Outcomes:CO1

Learning Domain with Level: Cognitive (Applying, Analyzing,Evaluating and Creating)

Submitted To

Rizoan Toufiq

Assistant Professor

Department of Computer Science and Engineering

Rajshahi University of Engineering and Technology

Submitted By

Name:Rafi Ahammed Songram

Roll:1803095

Department:Computer Science and Engineering

Rajshahi University of Engineering and Technology

K-Nearest Neighbor(KNN):

KNN algorithm is one of the most common classification algorithm based on supervised learning.It can be used for both regression and classification. But it is mostly used as a classification algorithm.
KNN is a non-parametric algorithm. That means it does not make any assumption or underlying data. It also called a lazy learner algorithm because it does not learn from the training data set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. It takes a data point and evaluates it with the existing data by calculating the distance. Then the algorithm assigns the new data point to a proper class by considering nearest neighbor. Here, distance can be Euclidean distance or Manhattan distance.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Steps of KNN Algorithm:

K-NN mainly assigns a new data point based on how closely it matches the points in the training set.
Step-1: At first the algorithm load the train set and test set.
Step-2: Select the number K which is the closest data points. Here K can be any positive integer. (The value of K is chosen such a way so that the accuracy is the maximum.)
Step-3: After choosing the value of K, the below steps are repeated:

- The distance between the test data point and train data points is measured. Here, distance can be Euclidean distance, Manhattan distance or Hamming distance. We used the Euclidean distance for our algorithm.
- After finding the distance, all the distances are sorted in ascending order.

- Then the algorithm will choose first K values.
- Finally, it will assign the test data point in a particular class which appears most in first K values.

Step-4: Repeat step-3 for each test data points.

Dataset:

The name of the dataset is **Diabetes**.The objective of the dataset is to diagnostically predict whether a patient has diabetes or not based on certain diagnostic measurements included in the dataset.The dataset includes 768 samples which is seperable from one another.From the data set in the (diabetes.csv) file We can find several variables, some of them are independent (several medical predictor variables) and only one target dependent variable (Outcome).
For train and test, we split the data set in two that is train and test .Test size =0.2. That means we took 768 different people data/samples. According to this 614 data points for training and 154 data points for testing from 768 data samples.

Table 1: Attribute of Dataset

Column	**Count**	**Type**
1 pregnencies	768	int64
2.Glucose	768	int64
3.BloodPressure	768	int64
4.SkinThickness	768	int64
5.Insulin	768	int64
6.BMI	768	float64
7.Diabetespedigrefunction	768	float64
8.Age	768	int64
9.Outcome	768	int64

Here are the all attribute of full datasets and according to this feature we apply KNN algorithm into this.

Characteristics plot of dataset:

Code-snippet

```
fig, ax = plt.subplots(4, 2, figsize=(10, 10))
for i in range(4):
for j in range(2):
feature = dataset.columns[i*2+j]
ax[i, j].hist(dataset[feature])
# ax[i, j].set_title(feature)
sk = skew(dataset[feature], bias=True)
#print(sk)
ax[i, j].set_xlabel(f'{feature} [Skewness={sk:.2f}]')
 ax[i, j].set_ylabel('Count')
fig.tight_layout()
```

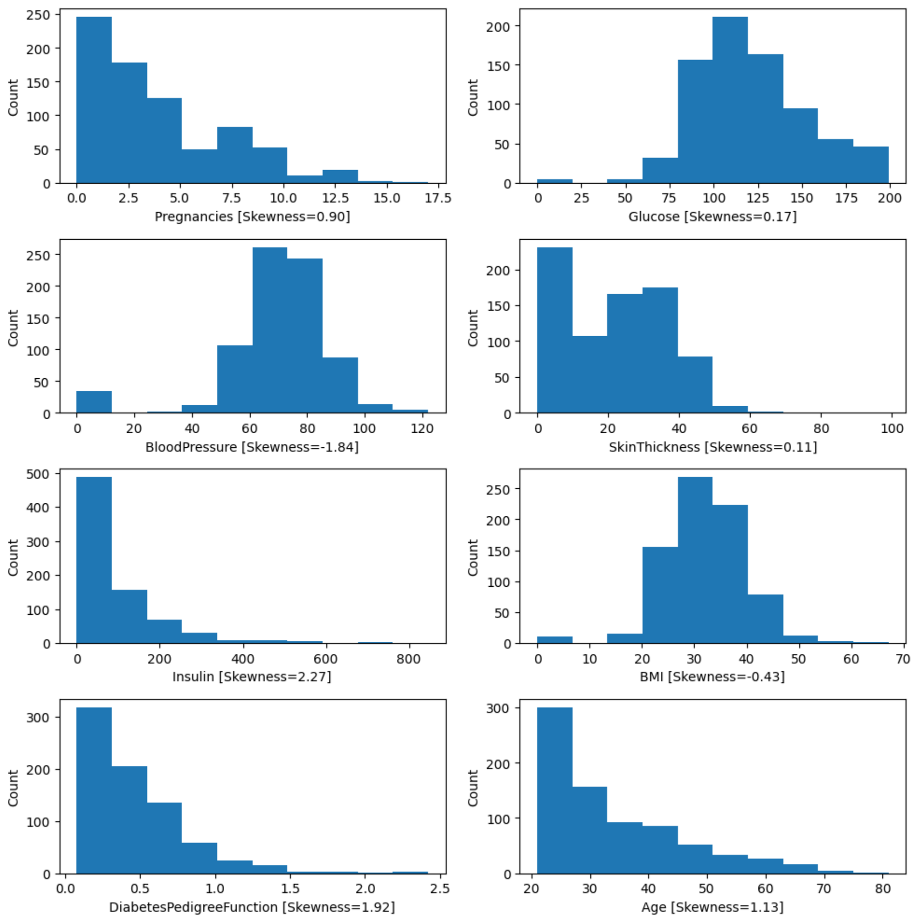

Figure-1: Characteristics Dataset-Feature Plot

Here some features 'Glucose', 'BloodPressure', 'SkinThickness', 'BMI' has some missing value that's why these features return outlair value that is not in skewed range[-1,1] and those features data are not in a uniformedly distributed manner.To solve this problem and to find a better accuracy from these dataset replace null value with mean and finally we got a uniformedly skewed distributed data plot.

Code-snippet

```
zero_not_accepted = ['Glucose', 'BloodPressure'
, 'SkinThickness', 'BMI']

for column in zero_not_accepted:
dataset[column] = dataset[column].replace(0, np.NaN)
mean = int(dataset[column].mean(skipna=True))
dataset[column] = dataset[column].replace(np.NaN, mean)


fig, ax = plt.subplots(4, 2, figsize=(10, 10))
for i in range(4):
for j in range(2):
feature = dataset.columns[i*2+j]
ax[i, j].hist(dataset[feature])
ax[i, j].set_title(feature)
sk = skew(dataset[feature], bias=True)
# print(sk)
ax[i, j].set_xlabel(f'{feature} [Skewness={sk:.2f}]')
ax[i, j].set_ylabel('Count')
fig.tight_layout()
```

Finally we overcome this problem and those features in a uniformed data and replace missing value with the mean value .The plot is given below

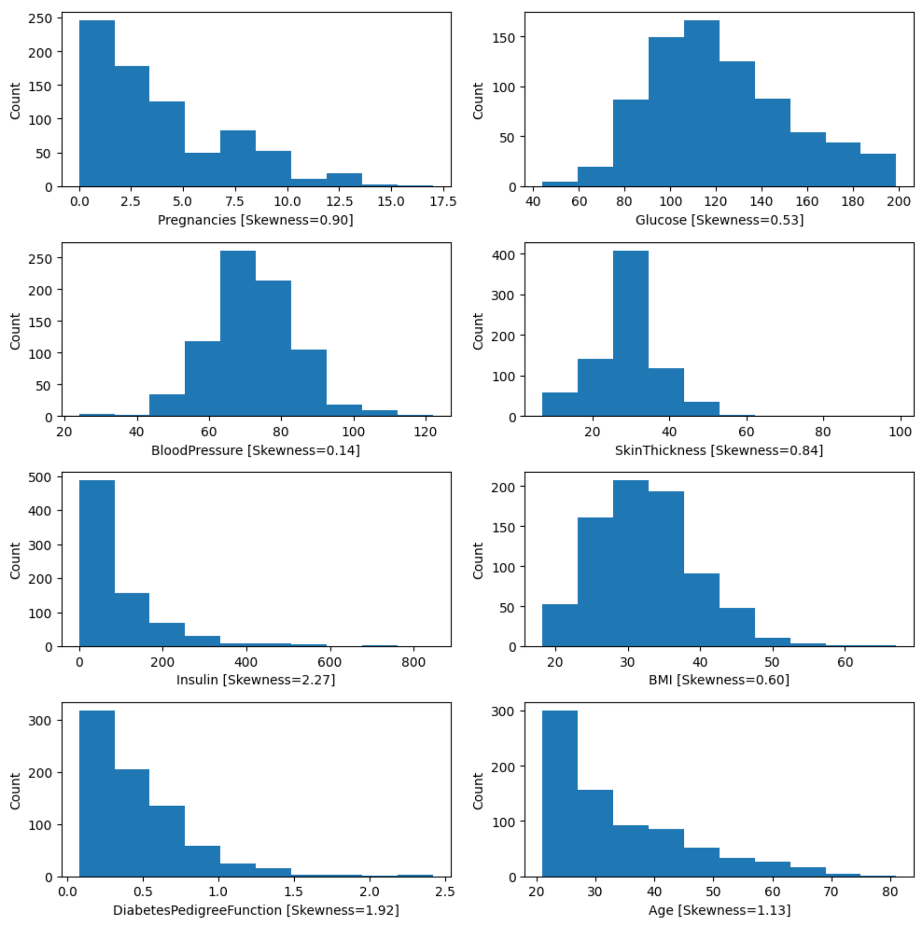

Figure-2: After Replacing Null with mean Dataset-Feature Plot

Split Dataset

Separates the features and target variable, and then splits them into training and testing sets. The training set (Xtrain and ytrain) is used to train a machine learning model, while the testing set (Xtest and ytest) is used to evaluate the model's performance.

```
X = dataset.iloc[:, 1:8]
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0, test_size=0.2)
```

Selecting columns 1 through 8 assuming that these columns contain the features for prediction. Eighth column (index 8) from the dataset, containing the target variable that we want to predict that is 1(diabetes-yes)and 0 (diabetes-No).

Feature Scaling

Feature scaling is a preprocessing step in machine learning that standardizes or normalizes the feature values to make them more suitable for certain machine learning algorithms. In this case, you're scaling the features in both the training and testing sets.

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Define Model and Selecting the value of K

Kneighborsclassifier is a class from scikit learn(a popular machine learning library) that implements the K-Nearest Neighbors algorithm.
K-NN classifier works basically two phase - Training and Testing and finally classification .
Training:During the training phase, the algorithm simply stores the entire dataset, including both the feature values and their corresponding class labels.
Testing: To make a prediction for a new data point, KNN looks for the k nearest data points in the training dataset.It measures the distance between the new data point and all the points in the training data.It then selects the k nearest data points the smallest one.
Classification:K-NN uses a majority and it counts the number of data points in each class among the k nearest neighbors and assigns the class label that occurs most frequently. This class label becomes the prediction for the new data point.

```
classifier = KNeighborsClassifier(n_neighbors=23,
p=2,metric='euclidean')
```

Nneighbors=23 it means the classifier will look at the 23 closest data points when making a prediction for a new data point.Selecting the right value of k in the k-Nearest Neighbors(KNN) algorithm is a crucial step.Using rules of thumb we use the square root of the number of data points math.sqrt(len(Xtrain) = 24.7790 as a starting point for k.We use the K value in a odd number and taking the value near which is 23. P=2 is for eucledian distance .

Model Fit and Evaluate Model

Evaluation of the understanding the KNN model performance on this dataset confusion matrix gives us a detailed overview of true positives,true negatives,false positives, and false negatives, which can be helpful for understanding clearly Code snippet and plot is given below.
Code-Snippet:

```
# Fit Model
classifier.fit(X_train, y_train)
#Evaluate Model
cm = confusion_matrix(y_test, y_pred)
print (cm)
plt.figure(figsize=(6, 4))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", cbar=False)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

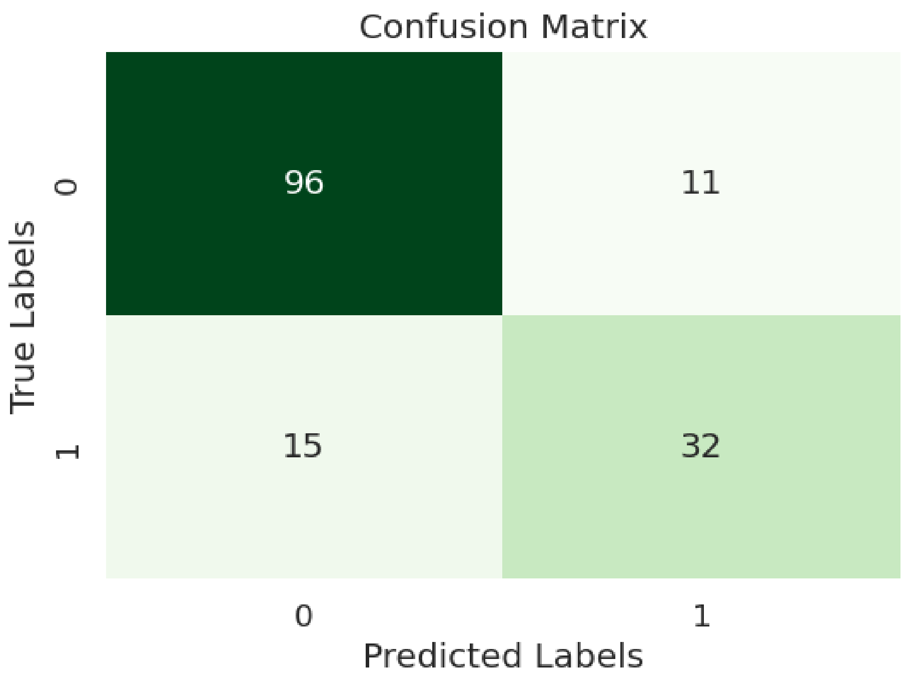

Figure-3:Confusion matrix

Accuracy

Code Snippet:

```
print(accuracy-score(y-test, y-pred))
```

Accuracy score function calculates the accuracy by comparing the true labels y-test with the predicted labels y-pred and then prints the accuracy value.The accuracy score is a value between 0 and 1.
The accuracy of this model = 83.5%

Train vs Test Accuracy

This train vs test accuracy plot visualize how the training and testing accuracy of a KNN model change according to varying the number of neighbor value-K.It can help to identify an optimal value of K, Results varying testing and training accuracy plot according to the values of K.By Increasing the value of k we get better test accuracy.
Code-Snippet:

```
k_values = list(range(1, 30))
train_accuracy = []
test_accuracy = []
for k in k_values:
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
train_accuracy.append(knn.score(X_train, y_train))
test_accuracy.append(knn.score(X_test, y_test))
plt.figure(figsize=(6, 6))
plt.plot(k_values, train_accuracy, marker='o', label='Train Accuracy')
plt.plot(k_values, test_accuracy, marker='o', label='Test Accuracy')
plt.title('KNN Train and Test Accuracy vs. K')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

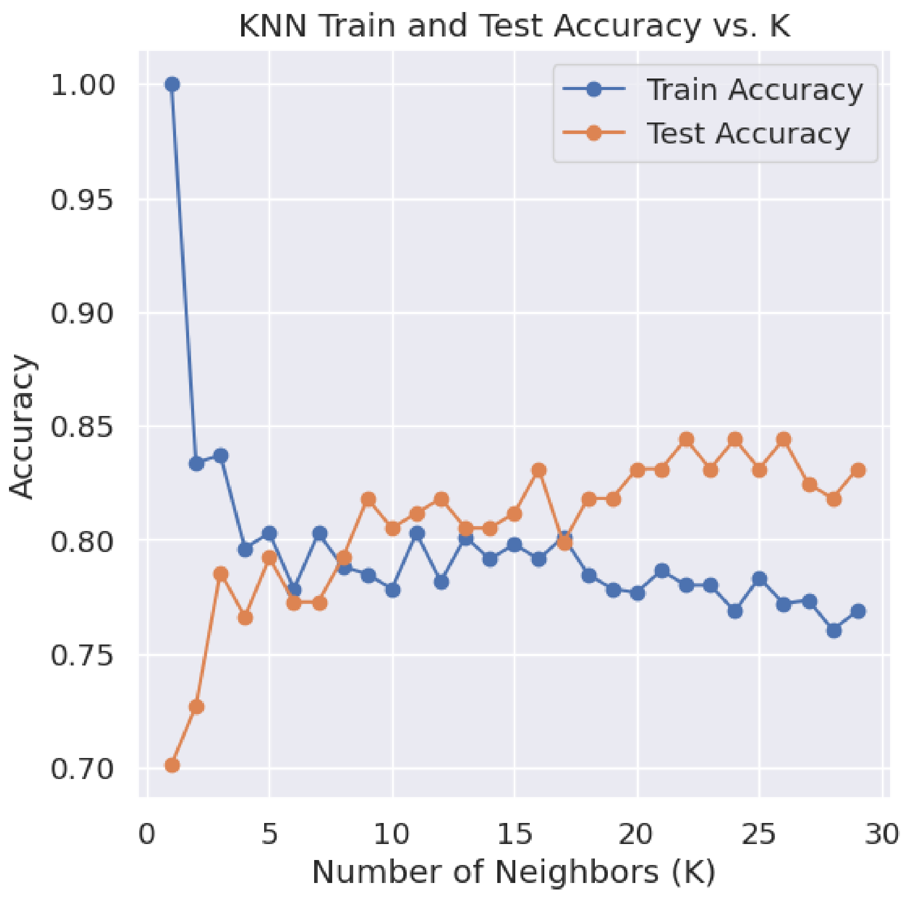

Figure-4:KNN Train vs Test Accuracy according to K

Diabetes Prediction according to Dataset

First calculate total count of people whose value is 0 with no diabetes and value 1 for those have diabetes.create a bar chart with two bars and labels "No" and "Yes.Count Total Yes and No,calculated separately and plotted.

```
fig, ax = plt.subplots(1, 1)
ax.set_xticks([0, 1])
ax.hist(y, label=['Outcome'])
ax.set_xlabel('outcome')
ax.set_ylabel('pc')
rects = ax.patches
labels = list(range(len(rects)))
labels[0] = 'No'
labels[-1] = 'Yes'
```

```
k = list(zip(rects, labels))
for rect, label in [k[0], k[-1]]:
height = rect.get_height()
ax.text(rect.get_x() + rect.get_width() / 2, height/2, label,
ha='center', va='bottom', color='white')
plt.show()
```

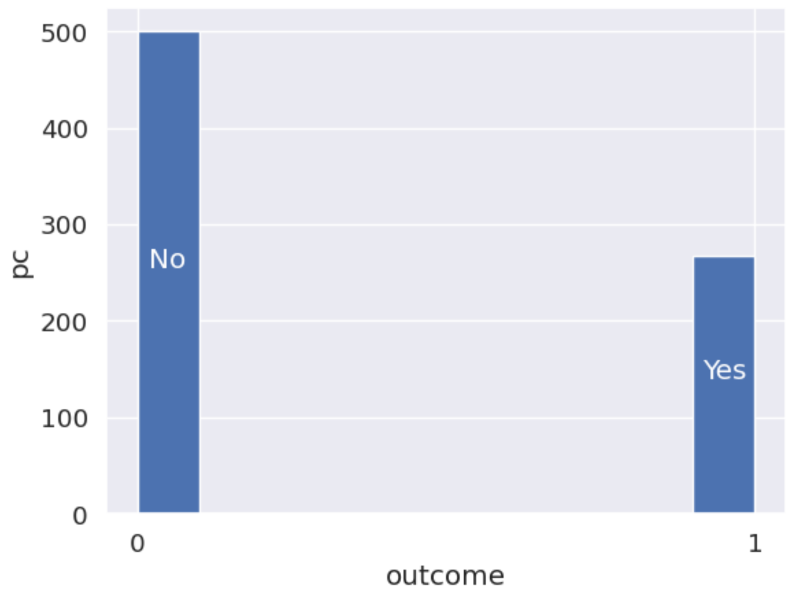

Figure-5:Plot according to Dataset

Diabetes prediction [KNN Model] Final Outcome

The bar chart is for y-test and y-pred .Blue bar is for y-test and orange bar is for y-pred.

```
fig, ax = plt.subplots(1, 1)
ax.set_xticks([0, 1])
ax.hist([y_test, y_pred], label=['y_test', 'y_pred'], bins=4)
ax.set_xlabel('outcome')
ax.set_ylabel('pc')
plt.legend()
rects = ax.patches
labels = list(range(len(rects)))
labels[0] = labels[4] = 'no'
labels[3] = labels[7] = 'yes'
```

```
k = list(zip(rects, labels))
for rect, label in k:
height = rect.get_height()
ax.text(rect.get_x() + rect.get_width() / 2, height/2, label,
ha='center', va='bottom', color='white')
plt.show()
```

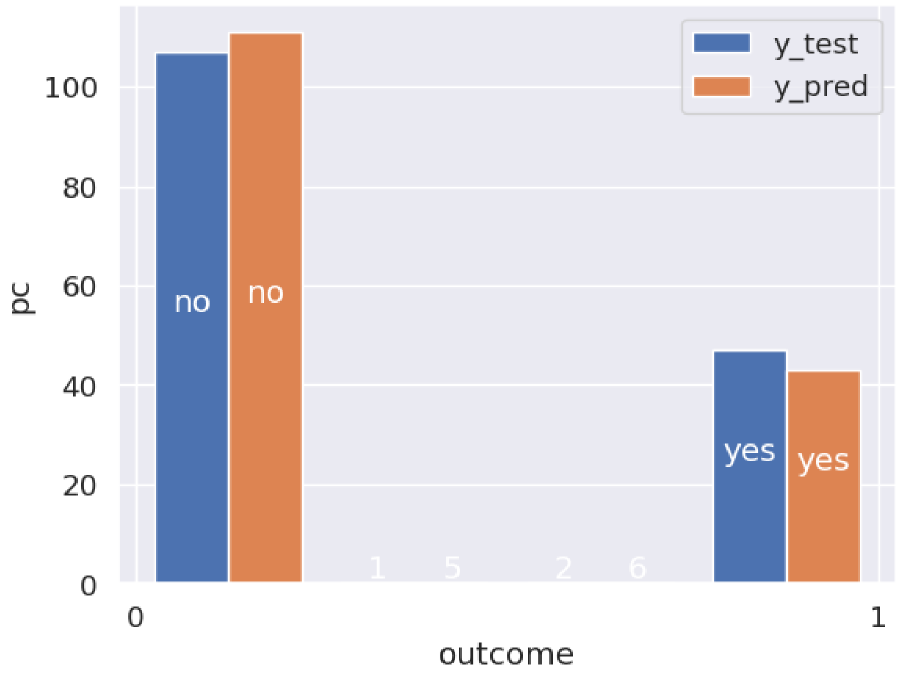

Figure-6:Final Outcome Applying KNN Model -Diabetes prediction

Limitations

The value of the K can be any integer. But we should consider some factors while choosing the value of K. For example, if K=1, then the calculation will be very easy. But the classification can be biased. Because, if the nearest neighbor is an outlier, then the classification won't be appropriate.A smaller K may lead to noise sensitivity, while a larger K may cause oversmoothing. Selecting the right K is often a trial and error process.

KNN can be computationally expensive, especially as the size of the dataset grows.It is not enough good and impractical on large dataset.

KNN may not handle datasets with a mix of data types (Numerical and Textual) well, as distance measures are not always straightforward to define in such cases.
If dataset contains a lot of noise or irrelevant features, KNN may not perform well, as it relies on the entire feature space for similarity calculation. or KNN model accuracy is very good but not great because we fit the model entire feature that's why accuracy fall down.

Improvements

K-Nearest Neighbors(KNN) algorithm can be improved and optimized using Feature Scaling,Distance matrics,Weighted KNN.
Feature Scalling:Feature scaling ensures that all features contribute equally to the distance calculation.It improves the accuracy.In the Diabetes dataset we applied feature scaling to get better accuracy.
Distance Metrics: Experiment with different distance metrics. The default metric in scikit-learn is the Euclidean distance,that we applied but other metrics like Manhattan, Mahalanobis distances might be more suitable for some datasets.
Weighted KNN: Assign different weights to each neighbor based on their distance. Closer neighbors can be given more weight, so their influence on the prediction is stronger.
If we could apply weighted techniques on our diabetes dataset the accuracy would get better.

Conclusion

Selecting the value of k using thumb rule and we select according to our data x-train and the k value is 23 .Iterate the k value and found test and train plot vs K. KNN is a very effective machine learning algorithm for classification .Handling outliers, class imbalances, and selecting an optimal K value are essential for improved performance.If the datasert is noisy then the accuracy might fall as our dataset is noisy and evaluate model with all features may decrease accuracy .By scalling and fine tuning the accuracy getting better.

References

[1] KNN:K Nearest-Neighbors by Micheal Steibach and Pang-Ning Tan

[2] A New Classwise k Nearest Neighbor(CKNN)Method for the Classification of Diabetes Dataset-(IJEAT)ISSN: $2249 - 8958$, Volume-2, Issue-3, February 2013