KENNESAW STATE
UNIVERSITY

CS 7267
MACHINE LEARNING

PROJECT 2

SUPERVISED LEARNING

INSTRUCTOR

Dr. Zongxing Xie

Job King
000618086

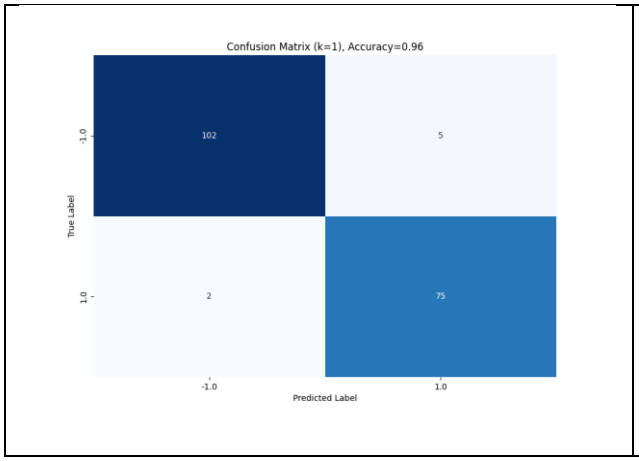
1. ABSTRACT

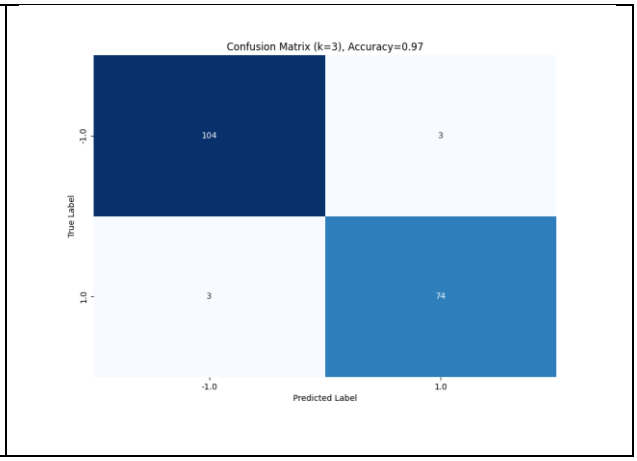
The purpose of this project was to implement the K-Nearest Neighbors machine learning algorithm and to evaluate the implementation by using it to analyze breast cancer data. The project steps included data normalization analysis of the dataset, splitting the dataset into training and testing sets, implementing the data distance calculation and data assignment portions of the algorithm and then finally to run several iterations of the breast cancer data against the algorithm. During the iterations, accuracy calculations and visualizations by way of a confusion matrix were performed to test the performance of the algorithm overall as well as to identify the best K-value.

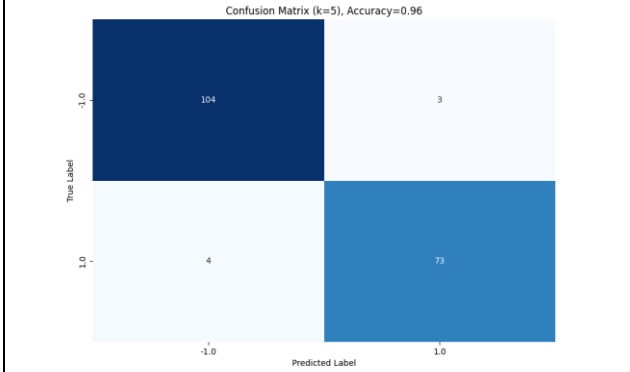
2. TEST RESULTS

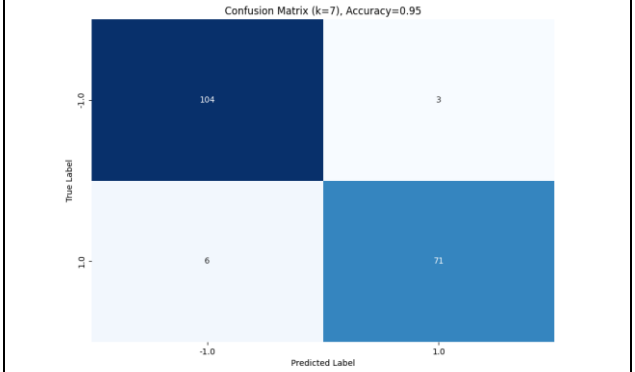
To determine whether the data needed to be analyzed, I performed simple statistical analysis on the feature data. For 20 rows, 2/3, the mean was less than 0.5. But for 2 rows, the mean was greater than 500. Those 2 clearly would have skewed the dataset, as the overall mean is 60.47. However, with a supermajority of the data having values under 0.5, even having 9 rows with means greater than 10 would skew the data. Further, 8 columns have means lower than 0.05 and 3 lower than 0.01. Normalization will be required to prevent both the columns with very large and very small means from producing less than ideal results. As min-max normalization is recommended for distance-based supervised learning algorithms (Geeks for Geeks, 2025) this was performed.

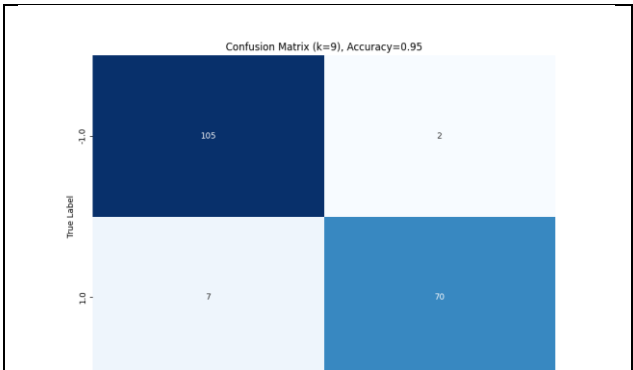
Afterwards, the completed program, which can be viewed in section 4, was used to analyze the normalized breast cancer data with k values 1, 3, 5, 7 and 9. Below are the confusion matrices for the tests along with their accuracy, in ascending order of k values. This shows that the accuracy peaks at 0.97 with k=3. Further the accuracy continues to decrease as K increases as a final test with k=100 shows.

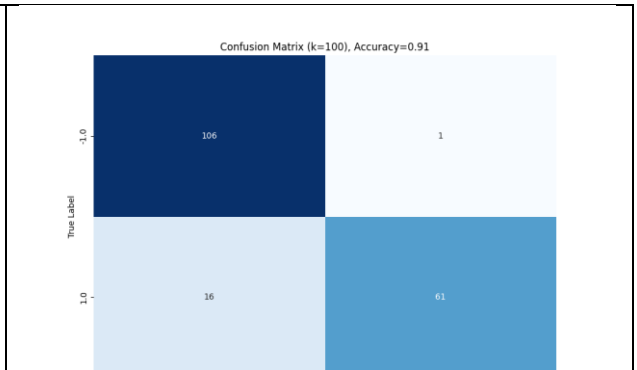












Further analysis shows that the false negatives increase with the K-value. With k=1, the false negative count was 2. With k being 3 for peak accuracy, there were 3 false negatives. With k=5 the false negative count was 4 and with k=9, the count was 7. For k=100, 16 false negatives were observed. With k=1000, whose matrix and accuracy score were not included there were 77 false negatives. Incidentally, as there were also 107 true negatives, there were no true or false positives.

3. DISCUSSION

I expected accuracy to improve with increasing k-values or at least to do so until some practical upper limit was reached. Instead, with the optimal k-value being reached so soon, on the third possible positive integer value, it seems that the optimal k-value for the data is determined by the data itself. It would be interesting to see if multiplying the label data, i.e. by 2 or 5 so that, for example, -2 would be negative or 5 for positive, would have an effect on the optimal k value. It would be curious to see if a different normalization method would produce a different optimal k value or even improve the accuracy. As logarithmic transformation addresses the primary issue with the data by compressing the large values and spreading out the small ones (Geeks for Geeks, 2025) it might have been a better solution for the data itself even if min-max normalization is commonly used for the kNN algorithm.

4. CODE

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix

def split_test_train(X, Y, training_set_val='70%', rand_seed=50):
training_set = float(training_set_val.strip('%')) / 100 #convert character percentage to float
total_rows = X.shape[0] #get the number of rows of feature data
train_size = int(training_set * total_rows) #calculate the number of training data rows of feature data
rng = np.random.default_rng(rand_seed) #initialize random number generator
sample_array_indices = rng.permutation(X.shape[0]) #generate random sample array of indices

train_array = sample_array_indices[0 : train_size] #from sample array indices splice the training set X
indices
test_array = sample_array_indices[train_size : total_rows] #from sample array indices splice the test set Y
indices
#recall that X and Y are same shape
return X[train_array], X[test_array], Y[train_array], Y[test_array]

def data_distances_calculation(X_train, X_test): #calculates the Euclidean distance
difference_distance = X_test[:, None, :] - X_train[None, :, :] #between the vectors that represent
sum_of_difference_distance_squared = np.sum(difference_distance ** 2, axis=2) #the training and test
data sets
return np.sqrt(sum_of_difference_distance_squared)

def assign_knn(k, y_train, calculated_distances):
#sort distances for each test sample, take indices of k nearest neighbors
neighbor_indices = np.argsort(calculated_distances, axis=1)[:, :k] # shape (X_test.shape[0], k)
predictions = []
for indices in neighbor_indices: #iterate over test sample neighbor indices
labels = y_train[indices] #pick labels of neighbors from training set
values, counts = np.unique(labels, return_counts=True) # find unique values in neighbor labels and how
many times they occur
max_index = np.argmax(counts) # find the index of the maximum number of times the unique value
occurs
predicted_label = values[max_index] # obtain the value from the index
predictions.append(predicted_label)
return np.array(predictions)

def knn_visualizer(predictions, Y_test, _k):
```

```
y_pred = np.array(predictions)
y_true = np.array(Y_test)
accuracy = accuracy_score(y_true, y_pred)
_confusion_matrix = confusion_matrix(y_true, y_pred)
labels = np.unique(y_true)

_df = pd.DataFrame(_confusion_matrix, index=labels, columns=labels)
plt.figure(figsize=(10,7))
sns.heatmap(_df, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title(f"Confusion Matrix (k={_k}), Accuracy={accuracy:.2f}")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
#plt.show()
plt.savefig(f"confusion_matrix-{_k}.png")

df = pd.read_csv("wdbc.data.mb.csv", header=None) #obtain entire dataframe
X_d = df.iloc[:, :-1].astype(float) #slice the first 30 rows as features dataframe
_Y = df.iloc[:, -1:].astype(float) #slice the last row as the labels dataframe
normalizer = MinMaxScaler() #initialize minmax normalizer
_X = pd.DataFrame(normalizer.fit_transform(X_d), columns=X_d.columns) #normalize the features
datafame
Y = _Y.to_numpy()
X = _X.to_numpy()

X_train, X_test, Y_train, Y_test = split_test_train(X, Y,'70%', 0)
calc_distances = data_distances_calculation(X_train, X_test)

k_values = [1, 3, 5, 7, 9]
for k in k_values:
knn_visualizer(assign_knn(k, Y_train, calc_distances), Y_test, k)
```

References

GeeksforGeeks. (2025, September 12). *Data Normalization Machine Learning*. https://www.geeksforgeeks.org/machine-learning/what-is-data-normalization/