CSCI4150U: Data Mining
Lab 04

Syed Naqvi
100590852

October 17, 2024

Preprocessing

German Credit Data (K-NN)

This dataset contains a mixture of ordinal, nominal and numeric features. The ordinal and nominal features must be processed such that Minkowski distance provides a meaningful metric during k nearest neighbors classifications while retaining as much information as possible. We begin with the following features which have either a completely arbitrary ordering or contain only 2 unique values:

- attribute 4: Purpose
  - a list of purchases on credit
- attribute 9: Personal status and sex
  - an arbitrarily ordered list of marital status and sex
- attribute 19: Telephone
  - either have a Telephone (yes) or do not (no)
- attribute 20: Foreign worker
  - either is a foreign worker (yes) or is not (no)

We can use **one-hot encoding** for Attributes 4 and 9 as these features have multiple unique values and **label encoding** for attributes 19 and 20 as these features have only two unique values which makes ordering of label encoding irrelevant.

```python
# first we can one-hot encode attribute 4 and 9
german_data_X_encodings = pd.get_dummies(data=german_data_X, columns=['Attribute4','Attribute9']).apply(lambda x: x.apply(lambda x: int(x) if isinstance(x,bool) else x))
display(german_data_X_encodings.loc[:,'Attribute4_A40':].head())
```

✓ 0.0s

	Attribute4_A40	Attribute4_A41	Attribute4_A410	Attribute4_A42	Attribute4_A43	Attribute4_A44	Attribute4_A45	Attribute4_A46	Attribute4_A48	Attribute4_A49	Attribute9_A91	Attribute9_A92	Attribute9_A93	Attribute9_A94
0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	0	0	0	1	0
4	1	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 1: [Attributes 4 and 9 one-hot encoding]

```python
# next we apply label encoding to attributes 19 and 20
label_encoder = LabelEncoder()
german_data_X_encodings[['Attribute19', 'Attribute20']] = german_data_X_encodings[['Attribute19', 'Attribute20']].apply(lambda x: label_encoder.fit_transform(x))
display(german_data_X_encodings[['Attribute19', 'Attribute20']].value_counts().reset_index(name='count'))
```

[31] ✓ 0.0s

...

	Attribute19	Attribute20	count
0	0	0	564
1	1	0	399
2	0	1	32
3	1	1	5

Figure 2: [Attributes 19 and 20 label encoding]

The next set of features appear to have a clear ordinal ranking based on descriptions provided at (https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data) with least credit-worthy values on the left to most credit-worthy values on the right:

- Attribute 1: Status of existing checking account
  - A11 (< 0 DM) → A12 (0 <= & < 200 DM) → A13 (>= 200 DM / salary assignments for at least 1 year) → A14 (no checking account)
- Attribute 3: Credit history
  - A34 (critical account / other credits existing) → A33 (delay in paying off in the past) → A32 (existing credits paid back duly till now) → A31 (all credits at this bank paid back duly) → A30 (no credits taken / all credits paid back duly)
- Attribute 6: Savings account / bonds
  - A61 (< 100 DM) → A62 (100 <= & < 500 DM) → A63 (500 <= & < 1000 DM) → A64 (>= 1000 DM) → A65 (unknown / no savings account)
- Attribute 7: Present employment since
  - A71 (unemployed) → A72 (< 1 year) → A73 (1 <= & < 4 years) → A74 (4 <= & < 7 years) → A75 (>= 7 years)
- Attribute 12: Property
  - A124 (unknown / no property) → A123 (car or other, not in attribute 6) → A122 (building society savings agreement / life insurance) → A121 (real estate)
- Attribute 17: Job
  - A171 (unemployed / unskilled - non-resident) → A172 (unskilled - resident) → A173 (skilled employee / official) → A174 (management / self-employed / highly qualified employee / officer)

We can visualize the value distributions of each feature for each class:

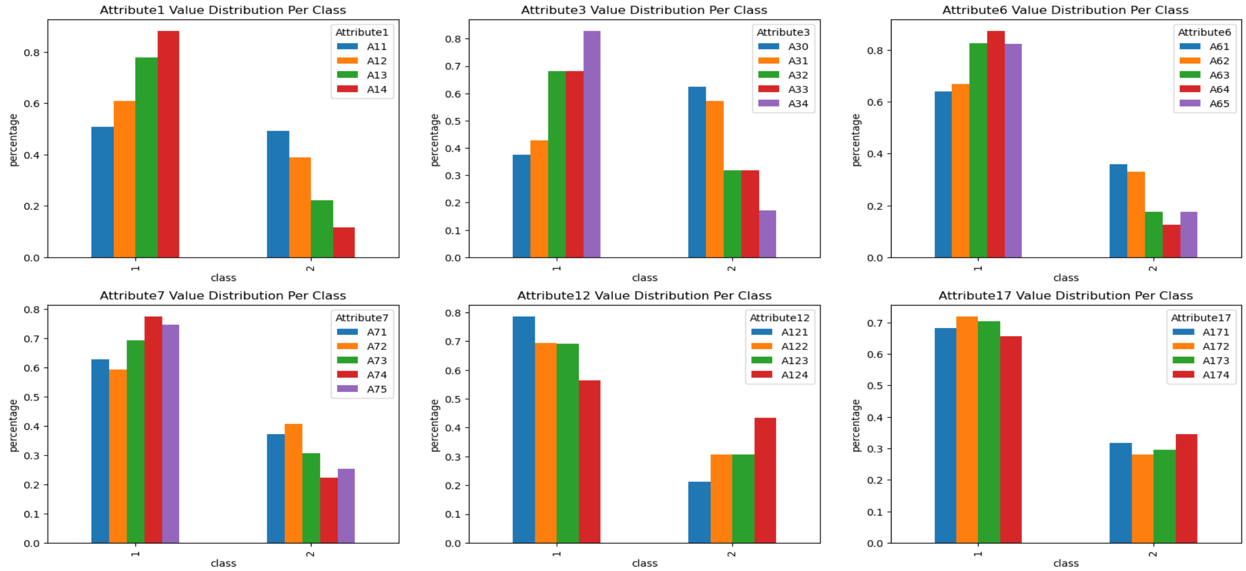


Figure 3: [Ordinal Features Visualization (Original Labeling)]

It can clearly be observed that the features do seem to closely adhere to their listed orderings and for features that do not, we can re-label them so their lexicographical order better fits the feature's observed order. This results in the following updated class distributions:

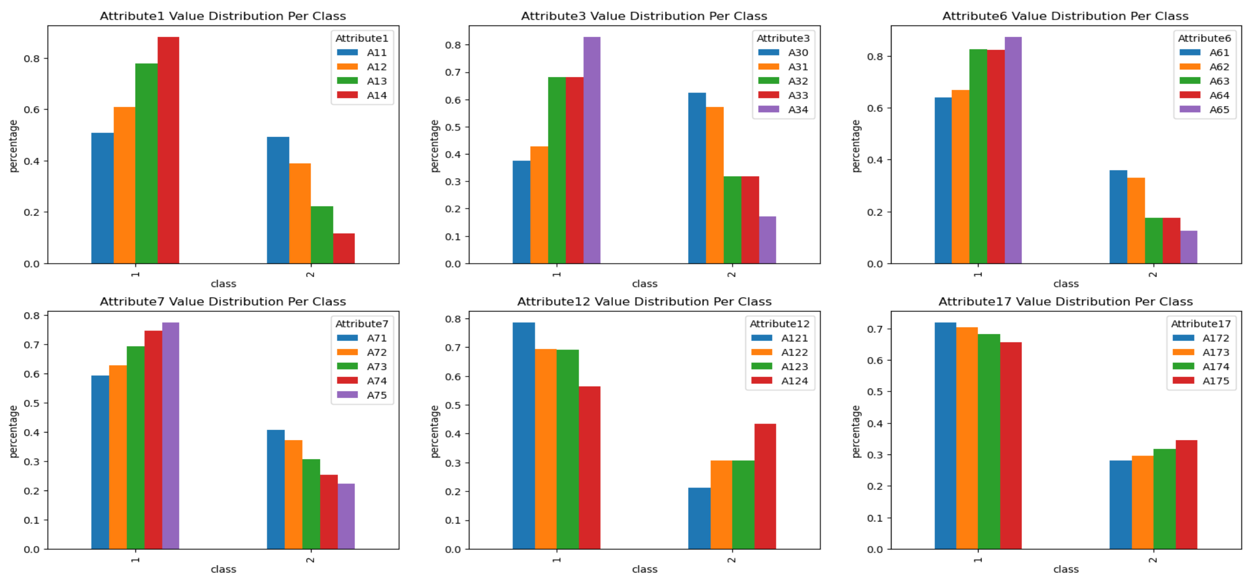


Figure 4: [Ordinal Features Visualization (Re-Labelled)]

Label encoding the above features should now result in labels that better capture feature order, allowing for more meaningful distance measures between objects during k-nearest neighbors classification.

```
# ordinal attributes label encoded
german_data_X_encodings[['Attribute1', 'Attribute3', 'Attribute6', 'Attribute7', 'Attribute12', 'Attribute17']].head()
```

✓ 0.0s

	Attribute1	Attribute3	Attribute6	Attribute7	Attribute12	Attribute17
0	0	4	3	3	0	1
1	1	2	0	2	0	1
2	3	4	0	4	0	0
3	0	2	0	4	1	1
4	0	3	0	2	3	1

Figure 5: [Ordinal Features Visualization (Re-Labelled)]

For the remaining qualitative features, we can use a similar plotting method as above to explore the existence of order. Based on the strength of potential orderings, we can either re-label values and then proceed with label encoding or apply one-hot encoding.

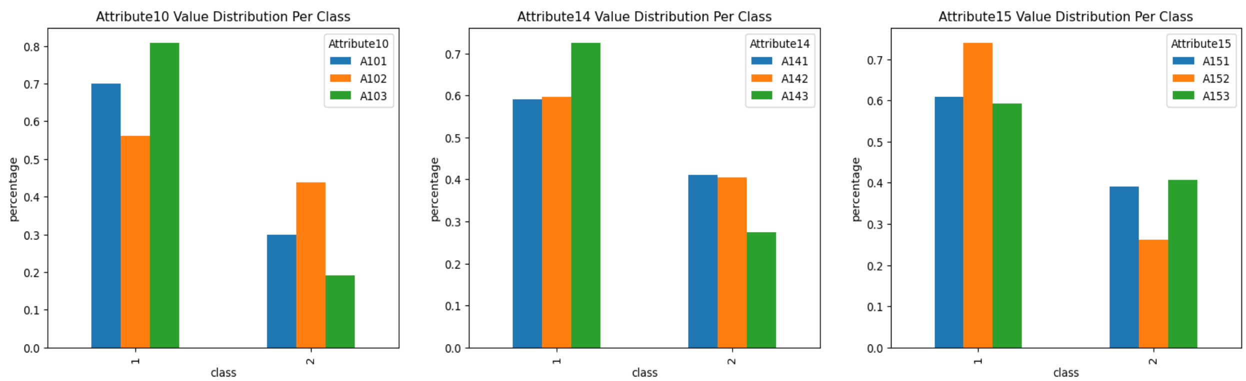


Figure 6: [Qualitative Features Visualization (Original Labeling)]

Only Attribute 10 seems to contain a useful order relation from least credit-worthy to most credit-worthy as follows:

- A102 (co-applicant) → A101 (no debtors/guarantors) → A103 (guarantor)

Attributes 14 and 15 do not seem to possess meaningful orderings as 2 out of 3 values for both features have an almost equal distribution across both classes. Hence, we shall re-label Attribute 10 to change its lexicographical order before applying label encoding and apply one-hot encoding to Attributes 14 and 15.

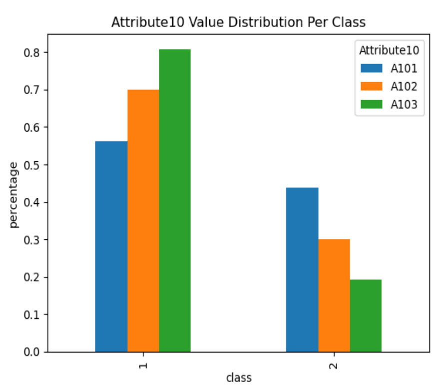


Figure 7: [Attribute 10 Visualization (Re-Labeled)]

```
# applying label encoding and one-hot encoding
german_data_X_encodings['Attribute10'] = label_encoder.fit_transform(german_data_X_encodings['Attribute10'])
german_data_X_encodings = pd.get_dummies(data=german_data_X_encodings,
                                         columns=['Attribute14', 'Attribute15']
                                         ).apply(
                                             lambda x: x.apply(lambda x: int(x) if isinstance(x,bool) else x)
                                             )
german_data_X_encodings.loc[:,['Attribute10'] + list(german_data_X_encodings.loc[:,'Attribute14_A141':].columns)].head()
```

[106] ✓ 0.0s

	Attribute10	Attribute14_A141	Attribute14_A142	Attribute14_A143	Attribute15_A151	Attribute15_A152	Attribute15_A153
0	1	0	0	1	0	1	0
1	1	0	0	1	0	1	0
2	1	0	0	1	0	1	0
3	2	0	0	1	0	0	1
4	1	0	0	1	0	0	1

Figure 8: [Attribute 10 Label Encoding, Attributes 14 & 15 One-Hot Encoding]

Finally, we standardize the numerical features of the dataset.

```python
standard_scaler = StandardScaler()
numerical_attributes = ['Attribute2', 'Attribute5', 'Attribute8', 'Attribute11', 'Attribute13', 'Attribute16', 'Attribute18']
german_data_X_encodings[numerical_attributes] = standard_scaler.fit_transform(X=german_data_X_encodings[numerical_attributes])
german_data_X_encodings[numerical_attributes].head()
```

[154] ✓ 0.0s

	Attribute2	Attribute5	Attribute8	Attribute11	Attribute13	Attribute16	Attribute18
0	-1.236478	-0.745131	0.918477	1.046987	2.766456	1.027079	-0.428290
1	2.248194	0.949817	-0.870183	-0.765977	-1.191404	-0.704926	-0.428290
2	-0.738668	-0.416562	-0.870183	0.140505	1.183312	-0.704926	2.334869
3	1.750384	1.634247	-0.870183	1.046987	0.831502	-0.704926	2.334869
4	0.256953	0.566664	0.024147	1.046987	1.535122	1.027079	2.334869

Figure 9: [Standardized Numerical Features]

German Credit Data (Decision Tree)

SciKit-Learn's decision tree classifier requires non-string feature values. As such, we apply label encoding to all qualitative features in the German Credit dataset.

```python
qualitative = ['Attribute1', 'Attribute3', 'Attribute4', 'Attribute6', 'Attribute7', 'Attribute9', 'Attribute10',
               'Attribute12', 'Attribute14', 'Attribute15', 'Attribute17', 'Attribute19', 'Attribute20',]
label_encoder = LabelEncoder()
german_data_X_tree = german_data_X.copy()
for q in qualitative:
    german_data_X_tree[q] = label_encoder.fit_transform(german_data_X_tree[q])

german_data_X_tree[qualitative].head()
```

[22] ✓ 0.0s

	Attribute1	Attribute3	Attribute4	Attribute6	Attribute7	Attribute9	Attribute10	Attribute12	Attribute14	Attribute15	Attribute17	Attribute19	Attribute20
0	0	4	4	4	4	2	0	0	2	1	2	1	0
1	1	2	4	0	2	1	0	0	2	1	2	0	0
2	3	4	7	0	3	2	0	0	2	1	1	0	0
3	0	2	3	0	3	2	2	1	2	2	2	0	0
4	0	3	0	0	2	2	0	3	2	2	2	0	0

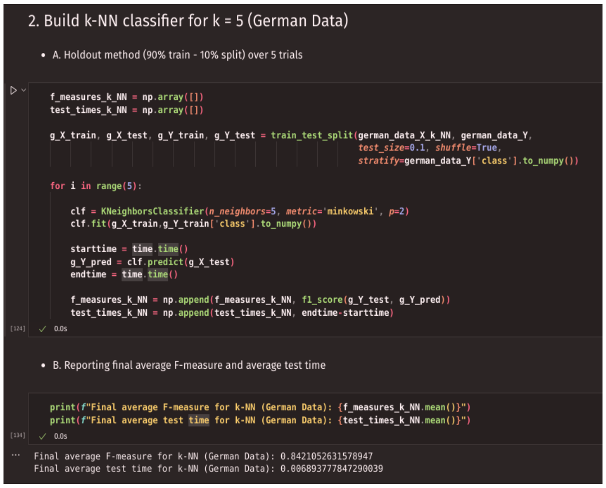
Figure 10: [Label Encoding German Data for Decision Tree Classification]

Waveform Data

All features of the waveform dataset are numeric by default with very similar scales making this dataset highly suitable for k-NN classification. The numeric data types are also compatible with scikit-learn's decision tree classifier. Hence, no preprocessing is required.

Part I (Inference Efficiency):

K-NN (k=5) and Decision Tree (Greedy, GINI, No Max Depth)



2. Build k-NN classifier for k = 5 (German Data)

- A. Holdout method (90% train - 10% split) over 5 trials

```python
f_measures_k_NN = np.array([])
test_times_k_NN = np.array([])

g_X_train, g_X_test, g_Y_train, g_Y_test = train_test_split(german_data_X_k_NN, german_data_Y,
                                                            test_size=0.1, shuffle=True,
                                                            stratify=german_data_Y['class'].to_numpy())

for i in range(5):

    clf = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
    clf.fit(g_X_train,g_Y_train['class'].to_numpy())

    starttime = time.time()
    g_Y_pred = clf.predict(g_X_test)
    endtime = time.time()

    f_measures_k_NN = np.append(f_measures_k_NN, f1_score(g_Y_test, g_Y_pred))
    test_times_k_NN = np.append(test_times_k_NN, endtime-starttime)
```

[124] ✓ 0.0s

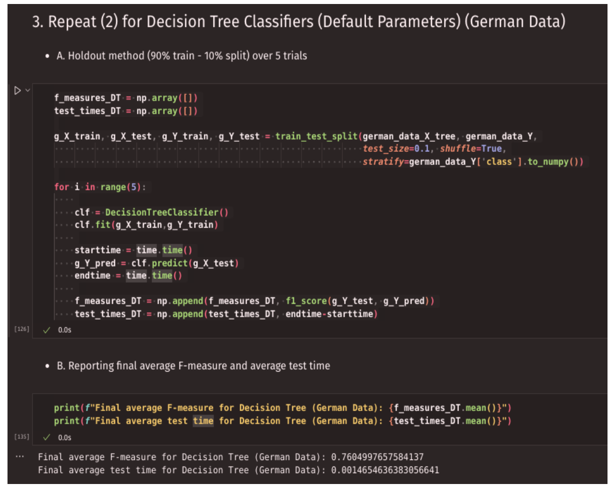
- B. Reporting final average F-measure and average test time

```python
print(f"Final average F-measure for k-NN (German Data): {f_measures_k_NN.mean()}")
print(f"Final average test time for k-NN (German Data): {test_times_k_NN.mean()}")
```

[134] ✓ 0.0s

```
Final average F-measure for k-NN (German Data): 0.8421052631578947
Final average test time for k-NN (German Data): 0.006893777847290039
```

Figure 11: [K-NN Classification for German Dataset]



3. Repeat (2) for Decision Tree Classifiers (Default Parameters) (German Data)

- A. Holdout method (90% train - 10% split) over 5 trials

```python
f_measures_DT = np.array([])
test_times_DT = np.array([])

g_X_train, g_X_test, g_Y_train, g_Y_test = train_test_split(german_data_X_tree, german_data_Y,
                                                            test_size=0.1, shuffle=True,
                                                            stratify=german_data_Y['class'].to_numpy())

for i in range(5):

    clf = DecisionTreeClassifier()
    clf.fit(g_X_train,g_Y_train)

    starttime = time.time()
    g_Y_pred = clf.predict(g_X_test)
    endtime = time.time()

    f_measures_DT = np.append(f_measures_DT, f1_score(g_Y_test, g_Y_pred))
    test_times_DT = np.append(test_times_DT, endtime-starttime)
```

[126] ✓ 0.0s

- B. Reporting final average F-measure and average test time

```python
print(f"Final average F-measure for Decision Tree (German Data): {f_measures_DT.mean()}")
print(f"Final average test time for Decision Tree (German Data): {test_times_DT.mean()}")
```

[135] ✓ 0.0s

```
Final average F-measure for Decision Tree (German Data): 0.7604997657584137
Final average test time for Decision Tree (German Data): 0.0014654636383056641
```

Figure 12: [Decision Tree Classification for German Dataset]

2. Build k-NN classifier for k = 5 (Waveform Data)

- A. Holdout method (90% train - 10% split) over 5 trials

```python
f_measures_k_NN_w = np.array([])
test_times_k_NN_w = np.array([])

w_X_train, w_X_test, w_Y_train, w_Y_test = train_test_split(waveform_data_X, waveform_data_Y,
                                                            test_size=0.1, shuffle=True)

for i in range(5):

    clf = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
    clf.fit(w_X_train,w_Y_train['class'].to_numpy())

    starttime = time.time()
    w_Y_pred = clf.predict(w_X_test)
    endtime = time.time()

    # since each class has a roughly equal number of instances, the 'macro' average argument is appropri
    f_measures_k_NN_w = np.append(f_measures_k_NN_w, f1_score(w_Y_test, w_Y_pred, average='macro'))

    test_times_k_NN_w = np.append(test_times_k_NN_w, endtime-starttime)
```

[128]

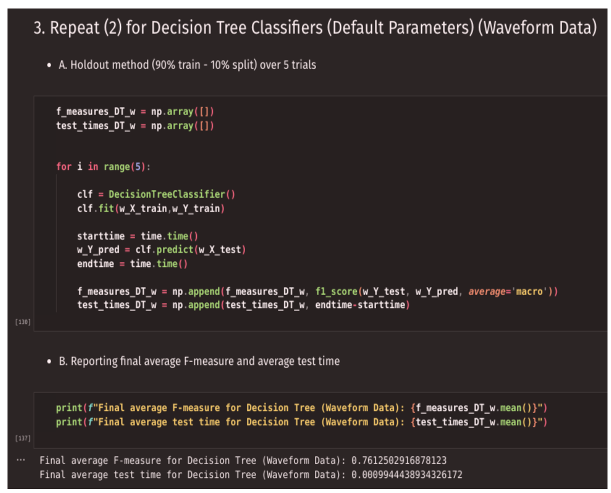
- B. Reporting final average F-measure and average test time

```python
print(f"Final average F-measure for k-NN (Waveform Data): {f_measures_k_NN_w.mean()}")
print(f"Final average test time for k-NN (Waveform Data): {test_times_k_NN_w.mean()}")
```

[136]

```
Final average F-measure for k-NN (Waveform Data): 0.8475760792639774
Final average test time for k-NN (Waveform Data): 0.02694072723388672
```

Figure 13: [K-NN Classification for Waveform Dataset]



3. Repeat (2) for Decision Tree Classifiers (Default Parameters) (Waveform Data)

- A. Holdout method (90% train - 10% split) over 5 trials

```python
f_measures_DT_w = np.array([])
test_times_DT_w = np.array([])


for i in range(5):

    clf = DecisionTreeClassifier()
    clf.fit(w_X_train,w_Y_train)

    starttime = time.time()
    w_Y_pred = clf.predict(w_X_test)
    endtime = time.time()

    f_measures_DT_w = np.append(f_measures_DT_w, f1_score(w_Y_test, w_Y_pred, average='macro'))
    test_times_DT_w = np.append(test_times_DT_w, endtime-starttime)
```

[130]

- B. Reporting final average F-measure and average test time

```python
print(f"Final average F-measure for Decision Tree (Waveform Data): {f_measures_DT_w.mean()}")
print(f"Final average test time for Decision Tree (Waveform Data): {test_times_DT_w.mean()}")
```

[137]

```
Final average F-measure for Decision Tree (Waveform Data): 0.7612502916878123
Final average test time for Decision Tree (Waveform Data): 0.0009944438934326172
```

Figure 14: [Decision Tree Classification for Waveform Dataset]

Comparing Results

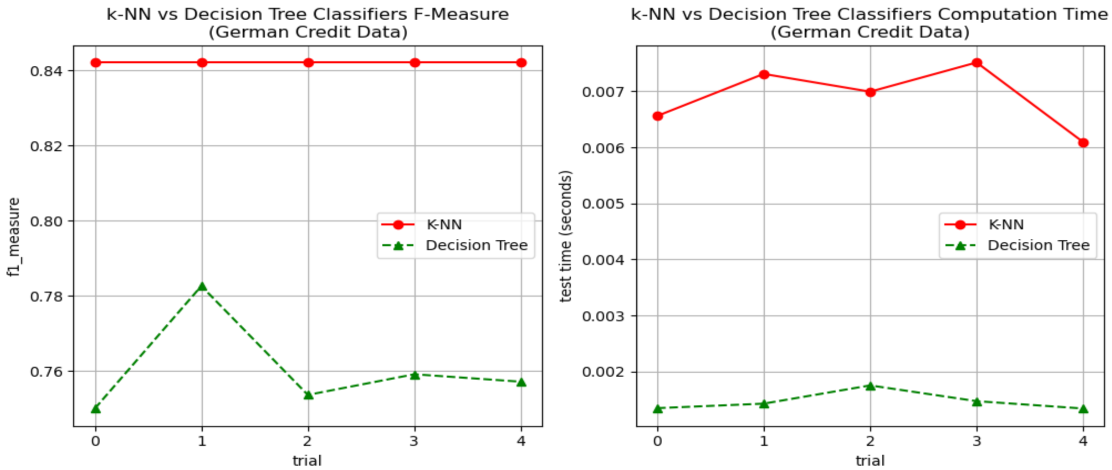


Figure 15: [K-NN vs Decision Tree Performance (German Dataset)]

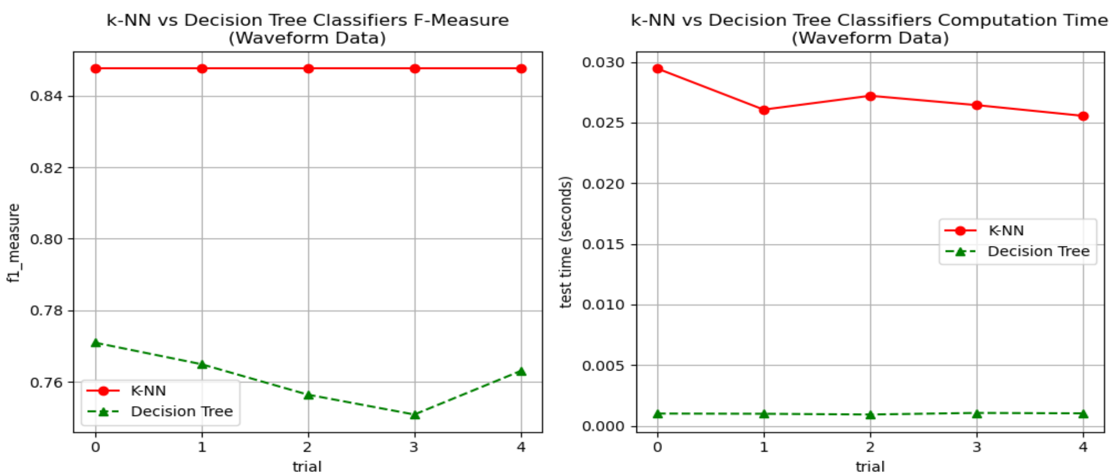


Figure 16: [K-NN vs Decision Tree Performance (Waveform Dataset)]

We see that k-NN is generally a more accurate form of classification than decision trees but also requires more elaborate preprocessing and greater inference times.

Part II: Model Selection

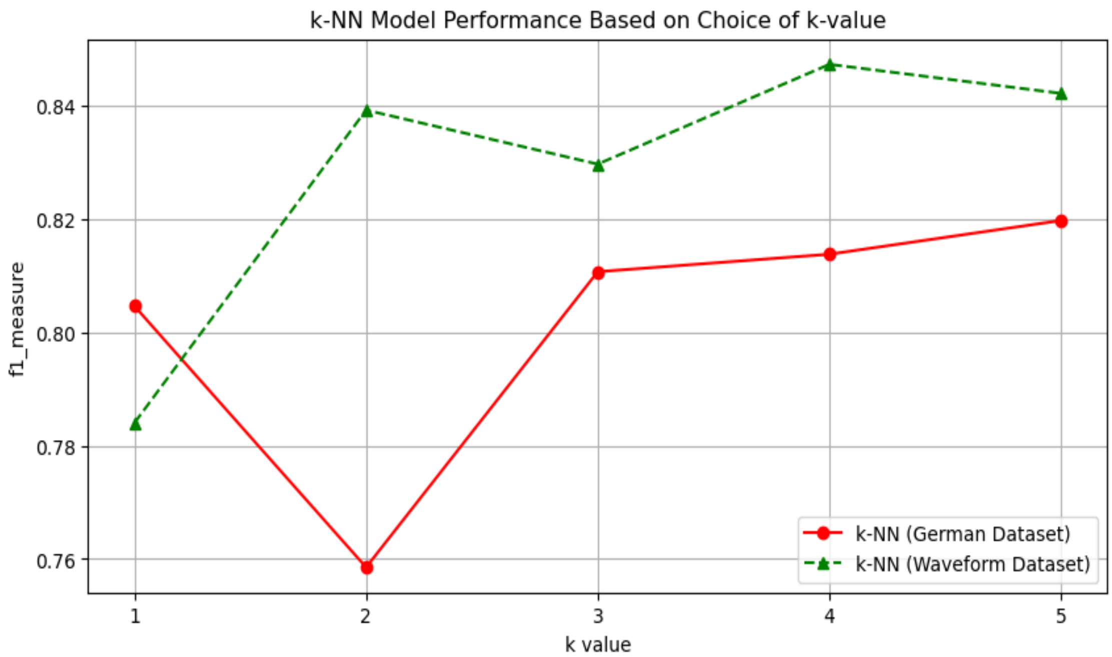


Figure 17: [K-NN Classification on Waveform and German Datasets for Different k]

Based on evaluations performed on the validation sets, the best k value for classifying Waveform data instances is k=4 and the best k value for classifying German credit data instances is k=5.

Figure 18: [Decision Tree Classification on Waveform and German Datasets with Different Max Depths]

Based on evaluations performed on the validation sets, the best max depth for classifying Waveform data instances is 7 and the best max depth for classifying German credit data instances is 6.