Lab 3 Classification Aidan Fitzsimmons

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas.api.types import is_numeric_dtype
from sklearn import preprocessing
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
import pydotplus
import io
from IPython.display import Image
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, f1_score, precision_score
import matplotlib.pyplot as pl
#Scale large numbers from 0=1
#predict the class
#
```

In [2]:
```python
german = pd.read_csv('german.data', header=None, sep = ' ')
wave = pd.read_csv('waveform.data')
```

In [3]:
```python
german.columns = ["Status of existing checking account",
                  "Duration in month",
                  "Credit history",
                  "Purpose",
                  "Credit amount",
                  "Savings account/bonds",
                  "Present employment since",
                  "Installment rate in percentage of disposable income",
                  "Personal status and sex",
                  "Other debtors / guarantors",
                  "Present residence since",
                  "Property",
                  "Age in years",
                  "Other installment plans",
                  "Housing",
                  "Number of existing credits at this bank",
                  "Job",
                  "Number of people being liable to provide maintenance for",
                  "Telephone",
                  "Foreign Worker",
                  "Class"
                 ]
```

In [4]:
```python
wave.columns = ["1",
                "2",
                "3",
                "4",
                "5",
                "6",
                "7",
                "8",
                "9",
                "10",
                "11",
                "12",
                "13",
                "14",
                "15",
                "16",
                "17",
                "18",
                "19",
                "20",
                "21",
                "Class"
               ]
```

In [5]:
```python
le = preprocessing.LabelEncoder()
for col in german.columns:
    if is_numeric_dtype(german[col]) == False:
        le.fit(german[col])
        german[col] = le.transform(german[col])
```

In [6]:
```python
#normalize the Credit Amount
german['Credit amount'] = MinMaxScaler().fit_transform(np.array(german['Credit amount']).reshape(-1,1))
german.head()
```

Out[6]:

	Status of existing checking account	Duration in month	Credit history	Purpose	Credit amount	Savings account/bonds	Present employment since	Installment rate in percentage of disposable income	Personal status and sex	Other debtors / guarantors	...	Property	Age in years
0	0	6	4	4	0.050567	4	4	4	2	0	...	0	67
1	1	48	2	4	0.313690	0	2	2	1	0	...	0	22
2	3	12	4	7	0.101574	0	3	2	2	0	...	0	49
3	0	42	2	3	0.419941	0	3	2	2	2	...	1	45
4	0	24	3	0	0.254209	0	2	3	2	0	...	3	53

5 rows × 21 columns

In [7]:
```python
Y = german['Class']
X = german.drop(['Class'],axis=1)
```

In [8]:
```python
Ywave = wave['Class']
Xwave = wave.drop(['Class'],axis=1)
```

Knn with n = 5 Minkowski Metrics

In [9]:
```python
testAcc = []
testf = []
testp = []

testAccwave = []
testfwave = []
testpwave = []

for i in range(5):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
    Xwave_train, Xwave_test, Ywave_train, Ywave_test = train_test_split(Xwave, Ywave, test_size=0.1)

    clf = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
    clf.fit(X_train, Y_train)

    clfwave = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
    clfwave.fit(Xwave_train, Ywave_train)

    Y_predTest = clf.predict(X_test)
    Ywave_predTest = clfwave.predict(Xwave_test)

    testAcc.append(accuracy_score(Y_test, Y_predTest))
    testf.append(f1_score(Y_test, Y_predTest, average='weighted'))
    testp.append(precision_score(Y_test, Y_predTest,average= 'weighted'))

    testAccwave.append(accuracy_score(Ywave_test, Ywave_predTest))
    testfwave.append(f1_score(Ywave_test, Ywave_predTest, average='weighted'))
    testpwave.append(precision_score(Ywave_test, Ywave_predTest,average= 'weighted'))
```

Default Parameters

In [10]:
```python
from sklearn import tree
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, f1_score, precision_score
#Need both sets
#Hold Out
testAccd = []
testfd = []
testpd = []

testAccwaved = []
testfwaved = []
testpwaved = []


for i in range(5):
    #split data
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
    Xwave_train, Xwave_test, Ywave_train, Ywave_test = train_test_split(Xwave, Ywave, test_size=0.1)
    #create n train model
    clf = tree.DecisionTreeClassifier()
    clfwave = tree.DecisionTreeClassifier()

    #features and labels
    clf.fit(X_train,Y_train)
    clfwave.fit(Xwave_train,Ywave_train)

    Y_predTest = clf.predict(X_test)
    Ywave_predTest = clfwave.predict(Xwave_test)

    testAccd.append(accuracy_score(Y_test, Y_predTest))
    testfd.append(f1_score(Y_test, Y_predTest, average='weighted'))
    testpd.append(precision_score(Y_test, Y_predTest,average= 'weighted'))

    testAccwaved.append(accuracy_score(Ywave_test, Ywave_predTest))
    testfwaved.append(f1_score(Ywave_test, Ywave_predTest, average='weighted'))
    testpwaved.append(precision_score(Ywave_test, Ywave_predTest,average= 'weighted'))
```

In [11]:
```python
finalAccC = np.mean(testAcc)
finalAccwaveC = np.mean(testAccwave)

finalfdC = np.mean(testf)
finalfdwaveC = np.mean(testfwave)

finalpC = np.mean(testp)
finalpwaveC = np.mean(testpwave)
```

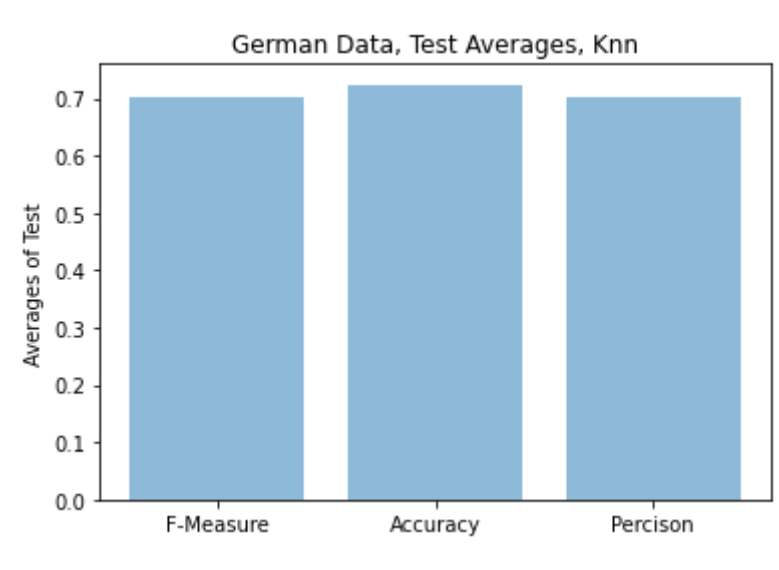
In [12]:
```python
finalAcc = np.mean(testAccd)
finalAccwave = np.mean(testAccwaved)

finalfd = np.mean(testfd)
finalfdwave = np.mean(testfwaved)

finalp = np.mean(testpd)
finalpwave = np.mean(testpwaved)
```

Compare

German Data

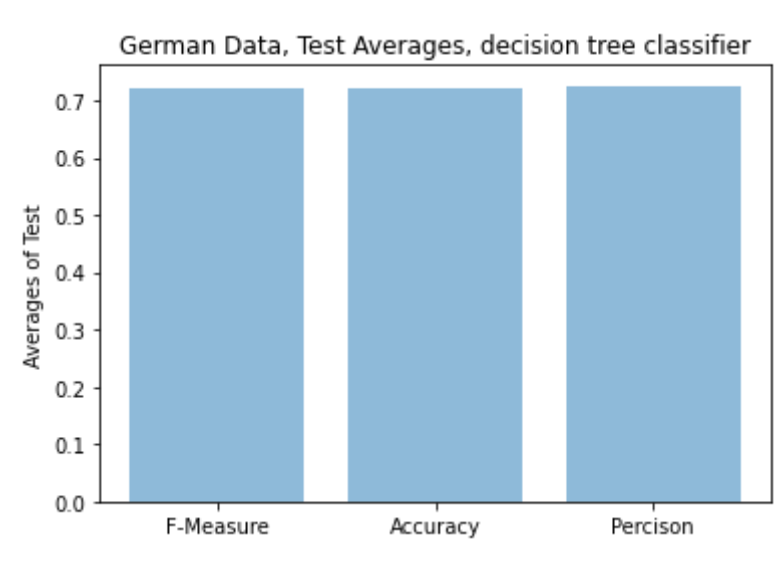
In [13]:
```python
objects = ('F-Measure', 'Accuracy', 'Percison')
y_pos = np.arange(len(objects))
performance = [finalfdC,finalAccC,finalpC]

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('German Data, Test Averages, Knn')
```

Out[13]: Text(0.5, 1.0, 'German Data, Test Averages, Knn')



In [14]:
```python
objects = ('F-Measure', 'Accuracy', 'Percison')
y_pos = np.arange(len(objects))
performance = [finalfd, finalAcc, finalp]

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('German Data, Test Averages, decision tree classifier')
```

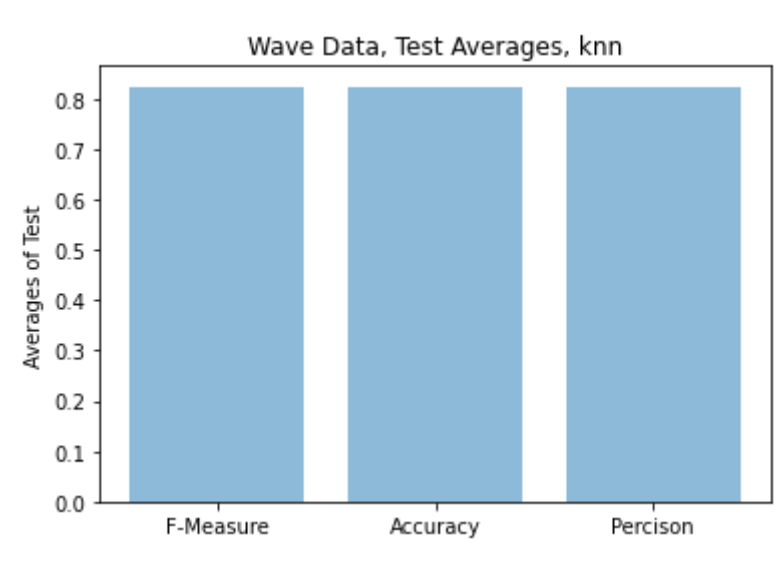
Out[14]: Text(0.5, 1.0, 'German Data, Test Averages, decision tree classifier')



Wave Data

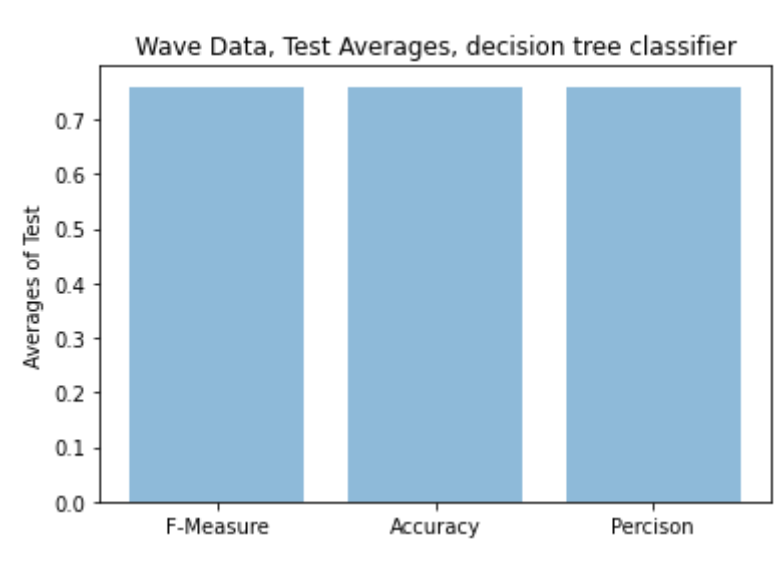
In [15]:
```python
objects = ('F-Measure', 'Accuracy', 'Percison')
y_pos = np.arange(len(objects))
performance = [finalfdwaveC,finalAccwaveC,finalpwaveC]

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('Wave Data, Test Averages, knn')
```

Out[15]: Text(0.5, 1.0, 'Wave Data, Test Averages, knn')



In [16]:
```python
objects = ('F-Measure', 'Accuracy', 'Percison')
y_pos = np.arange(len(objects))
performance = [finalfdwave, finalAccwave, finalpwave]

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('Wave Data, Test Averages, decision tree classifier')
```

Out[16]: Text(0.5, 1.0, 'Wave Data, Test Averages, decision tree classifier')



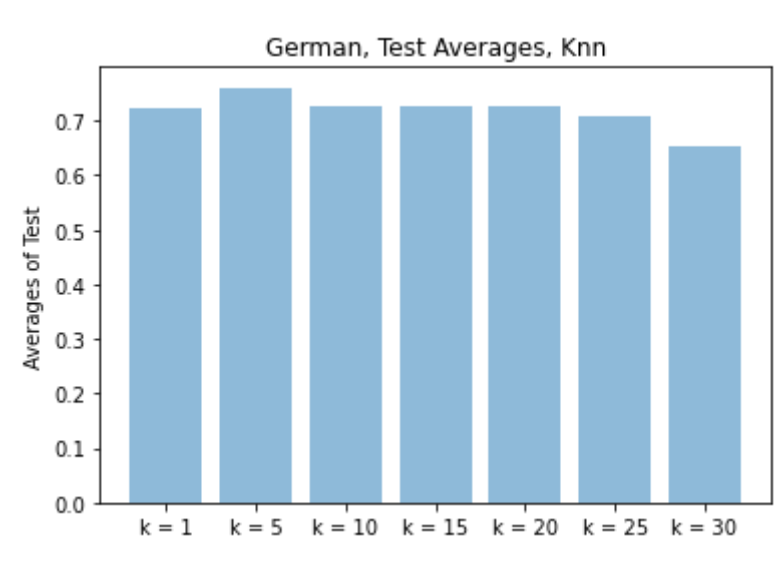
Model Selection

In [17]:
```python
numNeighbors = [1, 5, 10, 15, 20, 25, 30]

testf = []
testfwave = []

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
X_train, X_valid, Y_train, Y_valid = train_test_split(X_train, Y_train, train_size=0.9)

Xwave_train, Xwave_test, Ywave_train, Ywave_test = train_test_split(Xwave, Ywave, test_size=0.1)
Xwave_train, Xwave_valid, Ywave_train, Ywave_valid = train_test_split(Xwave_train, Ywave_train, train_size=

for k in numNeighbors:
    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
    clfwave = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)

    clf.fit(X_train, Y_train)
    clfwave.fit(Xwave_train, Ywave_train)

    Y_predTest = clf.predict(X_valid)
    Ywave_predTest = clfwave.predict(Xwave_valid)

    testf.append(f1_score(Y_valid, Y_predTest, average='weighted'))
    testfwave.append(f1_score(Ywave_valid, Ywave_predTest, average='weighted'))
```

The best model for german valid data, k = 15

In [18]:
```python
objects = ('k = 1', 'k = 5', 'k = 10','k = 15','k = 20','k = 25','k = 30',)
y_pos = np.arange(len(objects))
performance = testf

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('German, Test Averages, Knn')
```

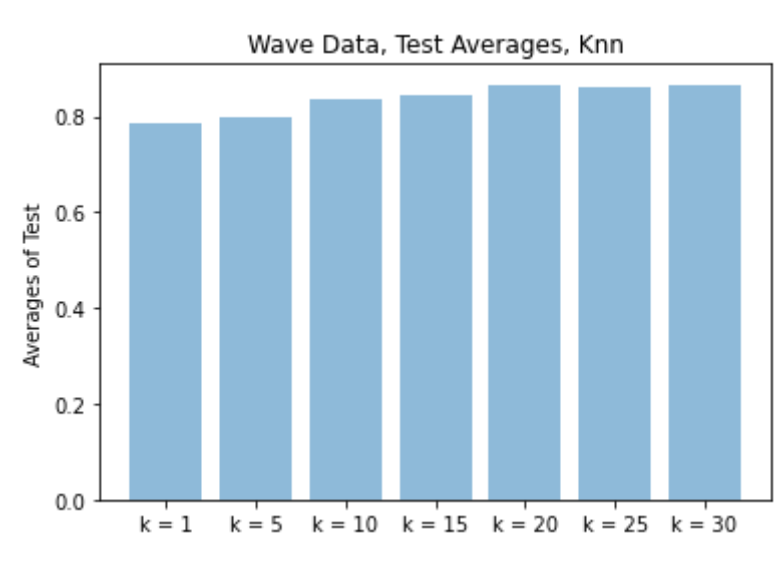
Out[18]: Text(0.5, 1.0, 'German, Test Averages, Knn')



The best model for wave valid data, k = 20

In [19]:
```python
objects = ('k = 1', 'k = 5', 'k = 10','k = 15','k = 20','k = 25','k = 30',)
y_pos = np.arange(len(objects))
performance = testfwave

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('Wave Data, Test Averages, Knn')
```

Out[19]: Text(0.5, 1.0, 'Wave Data, Test Averages, Knn')



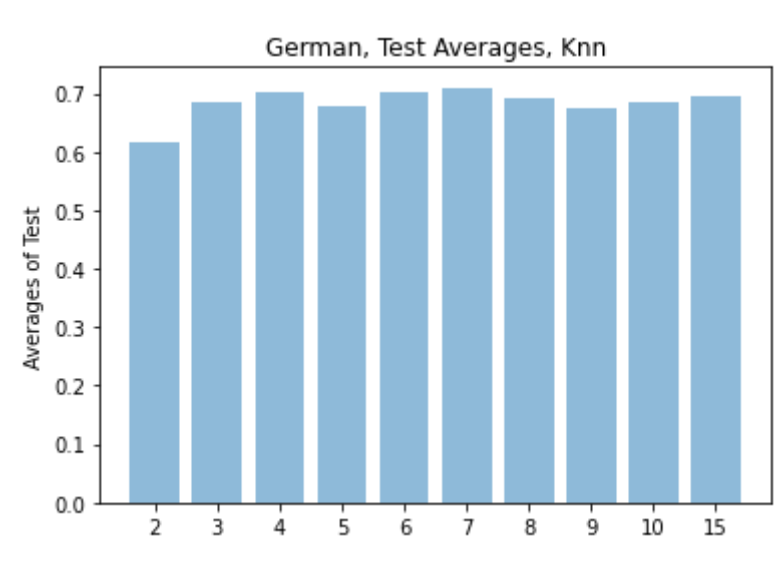
Validation on decision tree model

In [20]:
```python
###########################################
# Training and Test set creation
###########################################

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
X_train, X_valid, Y_train, Y_valid = train_test_split(X_train, Y_train, train_size=0.9)


Xwave_train, Xwave_test, Ywave_train, Ywave_test = train_test_split(Xwave, Ywave, test_size=0.1)
Xwave_train, Xwave_valid, Ywave_train, Ywave_valid = train_test_split(Xwave_train, Ywave_train, train_size=

from sklearn import tree
from sklearn.metrics import accuracy_score

###########################################
# Model fitting and evaluation
###########################################

maxdepths = [2,3,4,5,6,7,8,9,10,15]


validf = []
validfwave = []

trainAccwave = np.zeros(len(maxdepths))
index = 0
for depth in maxdepths:

    clf = tree.DecisionTreeClassifier(max_depth=depth)
    clf = clf.fit(X_train, Y_train)

    clfwave = tree.DecisionTreeClassifier(max_depth=depth)
    clfwave = clfwave.fit(Xwave_train, Ywave_train)

    Y_predTest = clf.predict(X_valid)
    Ywave_predTest = clfwave.predict(Xwave_valid)

    validf.append(f1_score(Y_valid, Y_predTest, average='weighted'))
    validfwave.append(f1_score(Ywave_valid, Ywave_predTest, average='weighted'))

    index += 1
```

Best Bin size for German data is 8

In [21]:
```python
objects = ('2','3','4','5','6','7','8','9','10','15')
y_pos = np.arange(len(objects))
performance = validf

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('German, Test Averages, Knn')
```

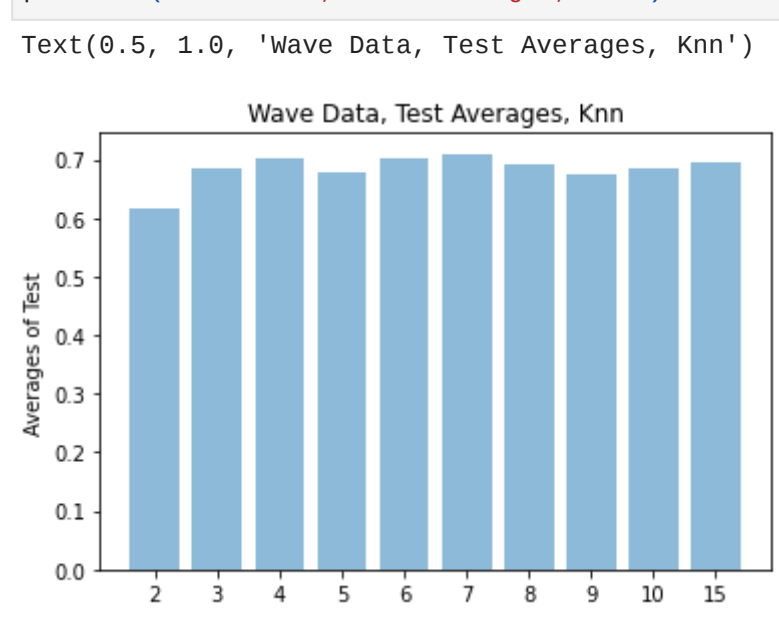
Out[21]: Text(0.5, 1.0, 'German, Test Averages, Knn')



Best bin size for Wave data is 8

In [22]:
```python
objects = ('2','3','4','5','6','7','8','9','10','15')
y_pos = np.arange(len(objects))
performance = validf

pl.bar(y_pos, performance, align='center', alpha=0.5)
pl.xticks(y_pos, objects)
pl.ylabel('Averages of Test')
pl.title('Wave Data, Test Averages, Knn')
```

Out[22]: Text(0.5, 1.0, 'Wave Data, Test Averages, Knn')



In [ ]: