COMP 6940 Assignment 3 811000293

In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
```

In [2]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import RandomizedSearchCV
```

In [3]:

```
from sklearn.metrics import recall_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, cross_val_score
```

In [4]:

```
#Q1: Loading and Cleaning Data
df = pd.read_csv('./data/data.csv',header=None)
```

In [5]:

```
df.shape
```

Out[5]:

```
(690, 16)
```

In [6]:

```
df.dtypes
```

Out[6]:

```
0      object
1      object
2     float64
3      object
4      object
5      object
6      object
7     float64
8      object
9      object
10      int64
11     object
12     object
13     object
14      int64
15     object
dtype: object
```

In [7]:

```
df.head()
```

Out[7]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b | 30.83 | 0.000 | u | g | w | v | 1.25 | t | t | 1 | f | g | 00202 | 0 | + |
| **1** | a | 58.67 | 4.460 | u | g | q | h | 3.04 | t | t | 6 | f | g | 00043 | 560 | + |
| **2** | a | 24.50 | 0.500 | u | g | q | h | 1.50 | t | f | 0 | f | g | 00280 | 824 | + |
| **3** | b | 27.83 | 1.540 | u | g | w | v | 3.75 | t | t | 5 | t | g | 00100 | 3 | + |
| **4** | b | 20.17 | 5.625 | u | g | w | v | 1.71 | t | f | 0 | f | s | 00120 | 0 | + |

In [8]:

```
#at a glance, the numbers in column 13 look like an id but they are not unique s
o we will treat them as numbers
print(len(set(df[13])),len(list(df[13])))
```

```
171 690
```

In [9]:

```
set(df[15]) #classes are clean
```

Out[9]:

```
{'+', '-'}
```

In [10]:

```
#convert classes/labels to integers.
#This should not be needed for decision
#trees but we will convert for later use
#with other algorithms
class_map = {'+':1,'-':-1}

df[15] = df[15].apply(lambda x: class_map[x])
```

In [11]:

```
df.head()
```

Out[11]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b | 30.83 | 0.000 | u | g | w | v | 1.25 | t | t | 1 | f | g | 00202 | 0 | 1 |
| **1** | a | 58.67 | 4.460 | u | g | q | h | 3.04 | t | t | 6 | f | g | 00043 | 560 | 1 |
| **2** | a | 24.50 | 0.500 | u | g | q | h | 1.50 | t | f | 0 | f | g | 00280 | 824 | 1 |
| **3** | b | 27.83 | 1.540 | u | g | w | v | 3.75 | t | t | 5 | t | g | 00100 | 3 | 1 |
| **4** | b | 20.17 | 5.625 | u | g | w | v | 1.71 | t | f | 0 | f | s | 00120 | 0 | 1 |

In [12]:

```
# If we encounter any non-numbers in our numerical columns, replace it with null
def numconv(x):
    try:
        return float(str(x))
    except Exception as e:
        return None
```

In [13]:

```
#For simplicity, I have chosen to identify which columns I want as numbers and c
ategories
nums = [1,2,7,10,13,14]
cats = [0,3,4,5,6,8,9,11,12]
```

In [14]:

```
#fill missing numbers with the mean value of the column
for num in nums:
    df[num] = df[num].apply(numconv)
    df[num] = df[num].fillna(df[num].mean())

df.head()
```

Out[14]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b | 30.83 | 0.000 | u | g | w | v | 1.25 | t | t | 1.0 | f | g | 202.0 | 0.0 | 1 |
| **1** | a | 58.67 | 4.460 | u | g | q | h | 3.04 | t | t | 6.0 | f | g | 43.0 | 560.0 | 1 |
| **2** | a | 24.50 | 0.500 | u | g | q | h | 1.50 | t | f | 0.0 | f | g | 280.0 | 824.0 | 1 |
| **3** | b | 27.83 | 1.540 | u | g | w | v | 3.75 | t | t | 5.0 | t | g | 100.0 | 3.0 | 1 |
| **4** | b | 20.17 | 5.625 | u | g | w | v | 1.71 | t | f | 0.0 | f | s | 120.0 | 0.0 | 1 |

In [15]:

```
for cat in cats:
    df[cat] = df[cat].apply(lambda x: str(x)).astype('category')
    print(cat,set(df[cat]))

df[cats].describe() #count imples that no nulls exist. We do however see questio
n marks. We will remove those rows
```

```
0 {'?', 'b', 'a'}
3 {'l', 'y', 'u', '?'}
4 {'g', 'gg', 'p', '?'}
5 {'j', 'ff', 'm', 'x', 'i', 'r', 'cc', 'k', 'q', 'c', 'aa', 'e',
'w', '?', 'd'}
6 {'j', 'ff', 'bb', 'n', 'v', 'h', 'z', '?', 'dd', 'o'}
8 {'f', 't'}
9 {'f', 't'}
11 {'f', 't'}
12 {'s', 'g', 'p'}
```

Out[15]:

| | 0 | 3 | 4 | 5 | 6 | 8 | 9 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 690 | 690 | 690 | 690 | 690 | 690 | 690 | 690 | 690 |
| **unique** | 3 | 4 | 4 | 15 | 10 | 2 | 2 | 2 | 3 |
| **top** | b | u | g | c | v | t | f | f | g |
| **freq** | 468 | 519 | 519 | 137 | 399 | 361 | 395 | 374 | 625 |

In [16]:

```
null_rows = df[((df[0] == '?') | (df[3] == '?') | (df[4] == '?') | (df[5] == '?'
) | (df[6] == '?'))]
print(null_rows.shape)
print(null_rows.index)
null_rows
```

```
(19, 16)
Int64Index([206, 248, 270, 327, 330, 346, 374, 453, 456, 479, 489, 5
20, 539,
            592, 598, 601, 622, 641, 673],
           dtype='int64')
```

Out[16]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **206** | a | 71.58 | 0.000 | ? | ? | ? | ? | 0.000 | f | f | 0.0 | f | p | 184.014771 | 0.0 | 1 |
| **248** | ? | 24.50 | 12.750 | u | g | c | bb | 4.750 | t | t | 2.0 | f | g | 73.000000 | 444.0 | 1 |
| **270** | b | 37.58 | 0.000 | ? | ? | ? | ? | 0.000 | f | f | 0.0 | f | p | 184.014771 | 0.0 | 1 |
| **327** | ? | 40.83 | 3.500 | u | g | i | bb | 0.500 | f | f | 0.0 | f | s | 1160.000000 | 0.0 | -1 |
| **330** | b | 20.42 | 0.000 | ? | ? | ? | ? | 0.000 | f | f | 0.0 | f | p | 184.014771 | 0.0 | -1 |
| **346** | ? | 32.25 | 1.500 | u | g | c | v | 0.250 | f | f | 0.0 | t | g | 372.000000 | 122.0 | -1 |
| **374** | ? | 28.17 | 0.585 | u | g | aa | v | 0.040 | f | f | 0.0 | f | g | 260.000000 | 1004.0 | -1 |
| **453** | ? | 29.75 | 0.665 | u | g | w | v | 0.250 | f | f | 0.0 | t | g | 300.000000 | 0.0 | -1 |
| **456** | b | 34.58 | 0.000 | ? | ? | ? | ? | 0.000 | f | f | 0.0 | f | p | 184.014771 | 0.0 | -1 |
| **479** | ? | 26.50 | 2.710 | y | p | ? | ? | 0.085 | f | f | 0.0 | f | s | 80.000000 | 0.0 | -1 |
| **489** | ? | 45.33 | 1.000 | u | g | q | v | 0.125 | f | f | 0.0 | t | g | 263.000000 | 0.0 | -1 |
| **520** | ? | 20.42 | 7.500 | u | g | k | v | 1.500 | t | t | 1.0 | f | g | 160.000000 | 234.0 | 1 |
| **539** | b | 80.25 | 5.500 | u | g | ? | ? | 0.540 | t | f | 0.0 | f | g | 0.000000 | 340.0 | -1 |
| **592** | b | 23.17 | 0.000 | ? | ? | ? | ? | 0.000 | f | f | 0.0 | f | p | 184.014771 | 0.0 | 1 |
| **598** | ? | 20.08 | 0.125 | u | g | q | v | 1.000 | f | t | 1.0 | f | g | 240.000000 | 768.0 | 1 |
| **601** | ? | 42.25 | 1.750 | y | p | ? | ? | 0.000 | f | f | 0.0 | t | g | 150.000000 | 1.0 | -1 |
| **622** | a | 25.58 | 0.000 | ? | ? | ? | ? | 0.000 | f | f | 0.0 | f | p | 184.014771 | 0.0 | 1 |
| **641** | ? | 33.17 | 2.250 | y | p | cc | v | 3.500 | f | f | 0.0 | t | g | 200.000000 | 141.0 | -1 |
| **673** | ? | 29.50 | 2.000 | y | p | e | h | 2.000 | f | f | 0.0 | f | g | 256.000000 | 17.0 | -1 |

In [17]:

```
df = df.drop(df.index[null_rows.index])
df = df.sample(frac=1)
df.reset_index(inplace=True,drop=True)

max(df.index)
```

Out[17]:

```
670
```

In [18]:

```
print(df.shape)
df.dtypes
```

```
(671, 16)
```

Out[18]:

```
0     category
1      float64
2      float64
3     category
4     category
5     category
6     category
7      float64
8     category
9     category
10     float64
11    category
12    category
13     float64
14     float64
15       int64
dtype: object
```

In [19]:

```
df[nums].describe()
```

Out[19]:

| | 1 | 2 | 7 | 10 | 13 | 14 |
|---|---|---|---|---|---|---|
| **count** | 671.000000 | 671.000000 | 671.000000 | 671.000000 | 671.000000 | 671.000000 |
| **mean** | 31.469639 | 4.831125 | 2.264694 | 2.461997 | 182.342926 | 1041.616990 |
| **std** | 11.705844 | 5.000660 | 3.377504 | 4.916433 | 169.892888 | 5281.226892 |
| **min** | 13.750000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 22.670000 | 1.040000 | 0.165000 | 0.000000 | 75.500000 | 0.000000 |
| **50%** | 28.580000 | 3.000000 | 1.000000 | 0.000000 | 160.000000 | 5.000000 |
| **75%** | 37.625000 | 7.500000 | 2.750000 | 3.000000 | 273.000000 | 400.000000 |
| **max** | 76.750000 | 28.000000 | 28.500000 | 67.000000 | 2000.000000 | 100000.000000 |

In [20]:

```
df.head()
```

Out[20]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | b | 18.83 | 0.415 | y | p | c | v | 0.165 | f | t | 1.0 | f | g | 200.0 | 1.0 | -1 |
| **1** | b | 31.08 | 1.500 | y | p | w | v | 0.040 | f | f | 0.0 | f | s | 160.0 | 0.0 | -1 |
| **2** | b | 37.75 | 7.000 | u | g | q | h | 11.500 | t | t | 7.0 | t | g | 300.0 | 5.0 | -1 |
| **3** | b | 21.83 | 0.250 | u | g | d | h | 0.665 | t | f | 0.0 | t | g | 0.0 | 0.0 | 1 |
| **4** | b | 39.17 | 1.625 | u | g | c | v | 1.500 | t | t | 10.0 | f | g | 186.0 | 4700.0 | 1 |

In [21]:

```
#encode categorical fields
for cat in cats:
    df[cat] = df[cat].cat.codes
df.head()
```

Out[21]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 18.83 | 0.415 | 3 | 3 | 2 | 8 | 0.165 | 0 | 1 | 1.0 | 0 | 0 | 200.0 | 1.0 | -1 |
| **1** | 2 | 31.08 | 1.500 | 3 | 3 | 13 | 8 | 0.040 | 0 | 0 | 0.0 | 0 | 2 | 160.0 | 0.0 | -1 |
| **2** | 2 | 37.75 | 7.000 | 2 | 1 | 11 | 4 | 11.500 | 1 | 1 | 7.0 | 1 | 0 | 300.0 | 5.0 | -1 |
| **3** | 2 | 21.83 | 0.250 | 2 | 1 | 4 | 4 | 0.665 | 1 | 0 | 0.0 | 1 | 0 | 0.0 | 0.0 | 1 |
| **4** | 2 | 39.17 | 1.625 | 2 | 1 | 2 | 8 | 1.500 | 1 | 1 | 10.0 | 0 | 0 | 186.0 | 4700.0 | 1 |

In [22]:

```
#separate our labels/classes from out input
X = df[list(range(0,15))]
Y = df[15]
(X.shape,Y.shape)
```

Out[22]:

```
((671, 15), (671,))
```

In [23]:

```
scaler = StandardScaler()
scaler.fit(X)
df_scale = scaler.transform(X)
df_scale = pd.DataFrame(df_scale)
df_scale.head()
```

Out[23]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.670257 | -1.080577 | -0.883767 | 1.780029 | 1.790785 | -1.114163 | 0.768204 | -0.62213 |
| **1** | 0.670257 | -0.033311 | -0.666634 | 1.780029 | 1.790785 | 1.457331 | 0.768204 | -0.65917 |
| **2** | 0.670257 | 0.536915 | 0.434041 | -0.543705 | -0.560715 | 0.989786 | -0.825196 | 2.736398 |
| **3** | 0.670257 | -0.824104 | -0.916787 | -0.543705 | -0.560715 | -0.646619 | -0.825196 | -0.47398 |
| **4** | 0.670257 | 0.658313 | -0.641619 | -0.543705 | -0.560715 | -1.114163 | 0.768204 | -0.22657 |

In [24]:

```
#explain 95% of variance
pca = PCA(0.95)
pca.fit(df_scale,Y)
print('# components:',pca.n_components_)
xpca = pca.transform(df_scale)
xpca = pd.DataFrame(xpca)
xpca.head()
```

```
# components: 13
```

Out[24]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| **0** | -2.186081 | 1.662291 | -1.286978 | 0.898303 | 1.095760 | 0.235137 | -1.036775 | -0.69578 |
| **1** | -2.942241 | 0.905070 | 1.123085 | -0.198643 | 1.305723 | -0.614909 | 2.260788 | -1.57239 |
| **2** | 2.835518 | 0.678521 | 1.881601 | 0.611875 | -0.358660 | -0.670573 | -0.167553 | -0.11403 |
| **3** | -0.341450 | -0.744306 | 0.230990 | -0.321905 | -0.119881 | -0.255688 | -0.912819 | 0.66205 |
| **4** | 1.593828 | -0.146370 | -0.724015 | 0.492527 | 1.217753 | 0.591135 | -1.377154 | -1.03930 |

In [25]:

```
X = xpca
(X.shape,Y.shape)
```

Out[25]:

```
((671, 13), (671,))
```

In [26]:

```
#Q2: Random Forest Classifier
#Part 1: Arbitrary/Default parameters
#while cross validation is not normally needed for this method,
#it is still used to obtain a recall value and compare against
#the OOB score.

import warnings
warnings.filterwarnings('ignore')

k_fold = KFold(n_splits=5)
k_fold.get_n_splits(X)

recall_scores = []
oobs = []

for train_idx,test_idx in k_fold.split(X):
    train_X, test_X = X.iloc[train_idx],X.iloc[test_idx]
    train_Y, test_Y = Y.iloc[train_idx],Y.iloc[test_idx]

    ran_forest = RandomForestClassifier(oob_score = True)
    ran_forest.fit(train_X, train_Y)
    ypred = ran_forest.predict(test_X)
    recall = round(recall_score(test_Y,ypred)*100,2)
    recall_scores.append(recall)
    oobs.append(ran_forest.oob_score_*100)

print('recall scores:',recall_scores)
print('avg recall score:',np.mean(recall_scores))
print('oob scores:',oobs)
print('avg oob score',np.mean(oobs))
```

```
recall scores: [75.0, 80.359999999999999, 72.730000000000004, 79.030
000000000001, 77.459999999999994]
avg recall score: 76.916
oob scores: [81.343283582089555, 75.791433891992554, 76.536312849162
016, 80.074487895716956, 82.495344506517682]
avg oob score 79.2481725451
```

In [27]:

```
#Q2: Random Forest Classifier
#Part 2: RandomizedSearchCV
ran_forest = RandomForestClassifier()

#set parameters to tune
params = {
    'n_estimators':list(range(1,11)),
    'max_features':list(range(1,14)),
    'max_depth':list(range(1,13)),
    'min_samples_split':list(range(2,11)),
    'min_samples_leaf':list(range(1,10)),
    'bootstrap': [True,False]
}
rscv = RandomizedSearchCV(ran_forest, n_iter=100, scoring='recall',\
                          param_distributions = params, \
                          refit=True,cv = 5,return_train_score=True)
rscv.fit(X,Y)
print('best recall score:', rscv.best_score_)
print('best parameters:', rscv.best_params_)
```

```
best recall score: 0.90998509687
best parameters: {'n_estimators': 2, 'min_samples_split': 7, 'min_sa
mples_leaf': 1, 'max_features': 12, 'max_depth': 2, 'bootstrap': Fal
se}
```

In [28]:

```
#Q3: KNN Classifier
#Part 1: Default Values
recall_scores = []

#using first from existing splits from above
for train_idx,test_idx in k_fold.split(X):
    train_X, test_X = X.iloc[train_idx],X.iloc[test_idx]
    train_Y, test_Y = Y.iloc[train_idx],Y.iloc[test_idx]

    knn = KNeighborsClassifier()
    knn.fit(train_X,train_Y)
    ypred = knn.predict(test_X)
    recall = round(recall_score(test_Y,ypred)*100,2)
    recall_scores.append(recall)
    break

print('recall score:',recall_scores)
```

```
recall score: [75.0]
```

In [29]:

```
#Q3: KNN CLassifier
#Part 2: Finding best value for k
neighbors = range(1,20,2)

cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors = k)
    scores = cross_val_score(knn,X,Y,cv=10,scoring='recall')
    cv_scores.append ({'val':k,'score':scores.mean()})
```

In [30]:

```
cv_scores = pd.DataFrame(cv_scores)
cv_scores.head()
```

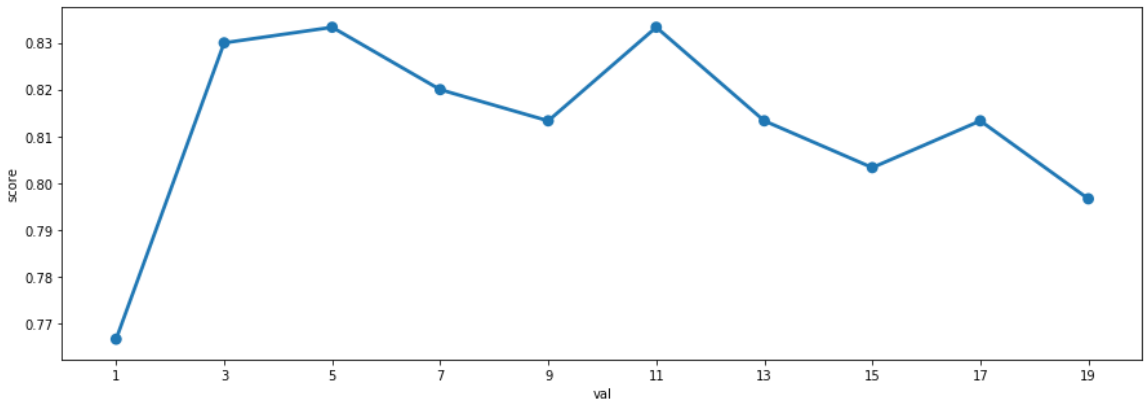
Out[30]:

| | score | val |
|---|---|---|
| **0** | 0.766667 | 1 |
| **1** | 0.830000 | 3 |
| **2** | 0.833333 | 5 |
| **3** | 0.820000 | 7 |
| **4** | 0.813333 | 9 |

In [31]:

```
plt.subplots(figsize=(15,5))
sns.pointplot(x='val',y='score',data=cv_scores)
print(cv_scores.shape)
plt.show()
```

```
(10, 2)
```



In [32]:

```
#the best cv recall score and its value for k
cv_scores = cv_scores.sort_values(by='score',ascending=False)
best_k = cv_scores.iloc[0,1]
print('best score:',cv_scores.iloc[0,0])
print('best k:',best_k)
```

```
best score: 0.833333333333
best k: 5
```

In [33]:

```
#determine the recall score of the classifier
#this shows that the recall score is still variable when given different trainin
g sets, even after shuffle.
kf = KFold(n_splits=5)
kf.get_n_splits(X)
recall_scores = []

for train_idx,test_idx in kf.split(X):
    train_X, test_X = X.iloc[train_idx],X.iloc[test_idx]
    train_Y, test_Y = Y.iloc[train_idx],Y.iloc[test_idx]

    knn = KNeighborsClassifier(n_neighbors=best_k)
    knn.fit(train_X,train_Y)
    ypred = knn.predict(test_X)
    recall = round(recall_score(test_Y,ypred)*100,2)
    recall_scores.append(recall)
    #break

print('recall score:',recall_scores)
```

```
recall score: [75.0, 87.5, 83.640000000000001, 85.480000000000004, 8
0.280000000000001]
```