1. Cross-validation for nearest neighbor classification.

In [1]:

```python
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
import matplotlib
from sklearn.preprocessing import MinMaxScaler
```

In [2]:

```python
dataframe = read_csv('wine.data', header = None, sep = ',')
dataframe
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

In [3]:

```python
# Separate features from labels
data = dataframe.values
X, y = data[:, 1:], data[:, 0]
```

(a) Use leave-one-out cross-validation (LOOCV) to estimate the accuracy of the classifier and also to estimate the 3 × 3 confusion matrix.

In [4]:

```python
loo = LeaveOneOut()
predict_values = []

for train_ix, test_ix in loo.split(X):
    X_train, X_test = X[train_ix, :], X[test_ix, :]
    y_train, y_test = y[train_ix], y[test_ix]
    neigh = KNeighborsClassifier(n_neighbors = 1)
    neigh.fit(X_train, y_train)
    predict_labels = neigh.predict(X_test)
    predict_values.append(predict_labels)
```

In [5]:

```python
predict_values = np.asarray(predict_values)
predict_values = predict_values.reshape(178,)

accuracy = accuracy_score(y, predict_values)
print("Accuracy rate:\n", accuracy)
```

```
Accuracy rate:
 0.7696629213483146
```

In [6]:

```python
print('Confusion matrix:\n', confusion_matrix(y, predict_values))
```

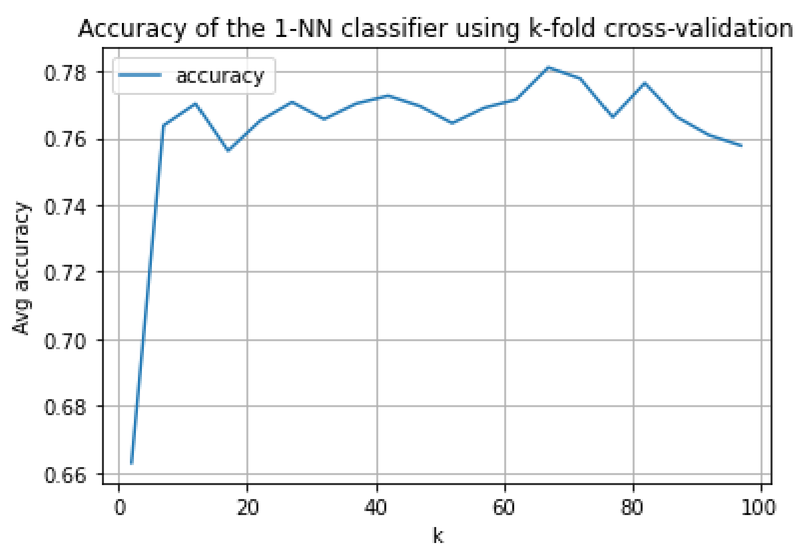
```
Confusion matrix:
 [[52  3  4]
 [ 5 54 12]
 [ 3 14 31]]
```

(b) Estimate the accuracy of the 1-NN classifier using k-fold cross-validation using 20 different choices of k that are fairly well spread out across the range 2 to 100. Plot these estimates: put k on the horizontal axis and accuracy estimate on the vertical axis.

In [7]:

```python
plot_data = []

for k in range(2, 100, 5):
    kf = KFold(n_splits = k, shuffle = True)
    acc_score = []

    for train_index, test_index in kf.split(X):
        kf_predict_values = []
        X_train_kf, X_test_kf = X[train_index, :], X[test_index, :]
        y_train_kf, y_test_kf = y[train_index], y[test_index]

        kf_neigh = KNeighborsClassifier(n_neighbors = 1)
        kf_neigh.fit(X_train_kf, y_train_kf)
        kf_predict_labels = kf_neigh.predict(X_test_kf)
        acc = accuracy_score(y_test_kf, kf_predict_labels)

        kf_predict_values.append(kf_predict_labels)
        acc_score.append(acc)

    avg_acc_score = sum(acc_score)/k
    plot_data.append(avg_acc_score)
    print('k=', k)
    print('\nAvg accuracy : {}\n'.format(avg_acc_score))

t = np.arange(2, 100, 5)
plt.xlabel('k')
plt.ylabel('Avg accuracy')
plt.title('Accuracy of the 1-NN classifier using k-fold cross-validation')
plt.plot(t, plot_data, label = 'accuracy')
plt.legend()
plt.grid()
plt.show()
```

```
k= 2

Avg accuracy : 0.6629213483146068

k= 7

Avg accuracy : 0.7637362637362637

k= 12

Avg accuracy : 0.7702380952380955

k= 17

Avg accuracy : 0.756149732620321

k= 22

Avg accuracy : 0.7651515151515152

k= 27

Avg accuracy : 0.7707231040564373

k= 32

Avg accuracy : 0.7656250000000003

k= 37

Avg accuracy : 0.7702702702702705

k= 42

Avg accuracy : 0.7726190476190476

k= 47

Avg accuracy : 0.7695035460992907

k= 52

Avg accuracy : 0.7644230769230769

k= 57

Avg accuracy : 0.7690058479532165

k= 62

Avg accuracy : 0.7715053763440861

k= 67

Avg accuracy : 0.7810945273631842

k= 72

Avg accuracy : 0.777777777777778

k= 77

Avg accuracy : 0.7662337662337663

k= 82

Avg accuracy : 0.7764227642276422

k= 87

Avg accuracy : 0.7662835249042145

k= 92

Avg accuracy : 0.7608695652173914

k= 97

Avg accuracy : 0.7577319587628866
```



(c) The various features in this data set have different ranges. Perhaps it would be better to normalize them so as to equalize their contributions to the distance function. There are many ways to do this; one option is to linearly rescale each coordinate so that the values lie in [0,1] (i.e. the minimum value on that coordinate maps to 0 and the maximum value maps to 1). Do this, and then re-estimate the accuracy and confusion matrix using LOOCV. Did the normalization help performance?

In [8]:

```python
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

In [9]:

```python
loo = LeaveOneOut()
predict_values = []
for train_ix, test_ix in loo.split(X):
    X_train, X_test = X[train_ix, :], X[test_ix, :]
    y_train, y_test = y[train_ix], y[test_ix]
    neigh = KNeighborsClassifier(n_neighbors = 1)
    neigh.fit(X_train, y_train)
    predict_labels = neigh.predict(X_test)
    predict_values.append(predict_labels)
```

In [10]:

```python
predict_values = np.asarray(predict_values)
predict_values = predict_values.reshape(178,)

acc = accuracy_score(y, predict_values)
print("\nAccuracy rate:\n", acc)
```

```
Accuracy rate:
 0.949438202247191
```

In [11]:

```python
print('Confusion matrix:\n', confusion_matrix(y, predict_values))
```

```
Confusion matrix:
 [[59  0  0]
 [ 5 62  4]
 [ 0  0 48]]
```

Yes, the normalization helps improve performance.

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

2. Binary logistic regression.

In [1]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.feature_selection import chi2
from sklearn.model_selection import cross_val_score
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [2]:

```
dataframe = read_csv('heart.csv')
dataframe
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

(a) Randomly partition the data into 200 training points and 103 test points. Fit a logistic regression model to the training data and display the coefficients of the model. If you had to choose the three features that were most influential in the model, what would they be?

In [3]:

```
# Separate features from labels
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 103/303, random_state = 42)
```

In [4]:

```
clf = LogisticRegression()
clf.fit(X_train, y_train)
predict = clf.predict(X_test)
coef = clf.coef_
print('Predicted values:\n', predict)
print('\nCoefficients of the model:\n', coef)
```

```
Predicted values:
 [0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1.
 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1.
 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0.
 1. 0. 1. 0. 1. 1. 1.]

Coefficients of the model:
 [[ 2.32276008e-02 -1.04450857e+00  9.11340974e-01 -8.11154500e-03
  -6.53211612e-04  5.32287897e-02  5.21463825e-01  1.87937460e-02
  -8.27825714e-01 -4.83269274e-01  7.53282739e-01 -1.37895276e+00
  -1.22092016e+00]]
```

In [5]:

```
# Three most influential features in this model
n_most_influential_features = 3
coef_three_features = np.argpartition(coef, X.shape[1] - n_most_influential_features)
coef_three_features = coef_three_features.tolist()

# Feature index
index_1 = coef_three_features[0][-1]
index_2 = coef_three_features[0][-2]
index_3 = coef_three_features[0][-3]

# Feature name
feature_1 = dataframe.columns[index_1]
feature_2 = dataframe.columns[index_2]
feature_3 = dataframe.columns[index_3]

print('Three most influential features in this model:\n', feature_1, feature_2, feature_3)
```

```
Three most influential features in this model:
 slope cp restecg
```

In [6]:

```
# Another way to select features is to use the chi2, find three smallest pvalues
n_most_influential_features_pvalue = 3
scores, pvalues = chi2(X_train, y_train)
three_features = np.argpartition(pvalues, n_most_influential_features_pvalue)

# Feature index
p_index_1 = three_features[0]
p_index_2 = three_features[1]
p_index_3 = three_features[2]

# Feature name
p_feature_1 = dataframe.columns[p_index_1]
p_feature_2 = dataframe.columns[p_index_2]
p_feature_3 = dataframe.columns[p_index_3]

print('Three most influential features in this model:\n', p_feature_1, p_feature_2, p_feature_3)
```

```
Three most influential features in this model:
 thalach ca cp
```

(b) What is the test error of your model?

In [7]:

```
accuracy = accuracy_score(y_test,predict)
error = 1 - accuracy
print('The test error of the model is:\n', error)
```

```
The test error of the model is:
 0.19417475728155342
```

(c) Estimate the error by using 5-fold cross-validation on the training set. How does this compare to the test error?

In [8]:

```
kf = KFold(n_splits = 5, random_state = 42, shuffle = True)
acc_score = []

for train_index, test_index in kf.split(X):
    kf_predict_values = []
    X_train_kf, X_test_kf = X[train_index, :], X[test_index, :]
    y_train_kf, y_test_kf = y[train_index], y[test_index]

    clf = LogisticRegression(solver = 'liblinear')
    clf.fit(X_train_kf, y_train_kf)
    kf_predict_labels = clf.predict(X_test_kf)
    acc = accuracy_score(y_test_kf, kf_predict_labels)
    kf_predict_values.append(kf_predict_labels)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/5
error_kf = 1-avg_acc_score

print('\nAvg accuracy : {}'.format(avg_acc_score))
print('\nError rate : {}'.format(error_kf))
```

```
Avg accuracy : 0.8282513661202187

Error rate : 0.17174863387978134
```

This error rate is a little bit lower than the pure logistic regression classifier. The 5-fold cross-validation has a higher accuracy than the pure logistic regression classifier.

In [ ]:

3. Stepwise forward selection.

In [1]:
```python
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
import pandas as pd
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score as acc
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.model_selection import cross_val_score
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [2]:
```python
dataframe = read_csv('heart.csv')
dataframe
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [3]:
```python
# Separate features from labels
data = dataframe.values
X, y = data[:, :-1], data[:, -1]

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 103/303, random_state = 42)
```

(a) Use this procedure to find a k-sparse logistic regression solution for the heart disease data, for k = 1, 2, . . . , 13. Create a single plot showing the test error and cross-validation error for all these values of k.

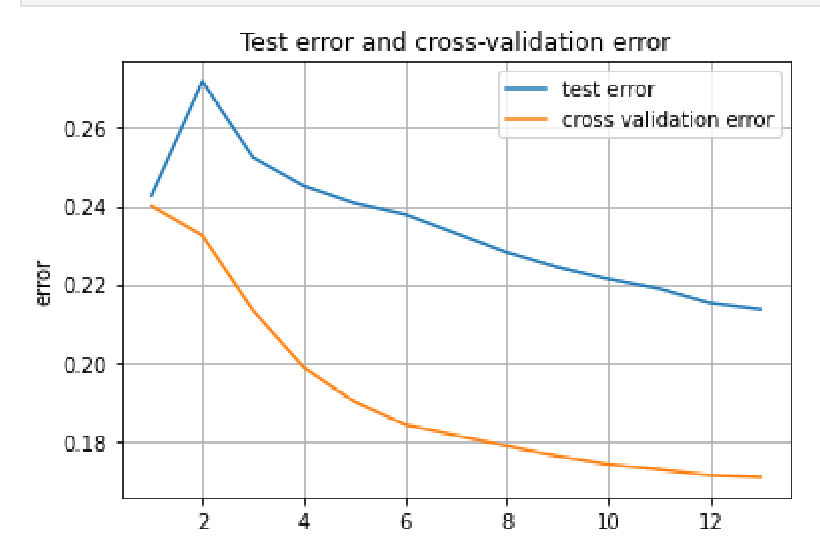
In [4]:
```python
# FIRST OPTION: hand written stepwise forward selection

def sfs_function(x_train_data, y_train_data, x_test_data, y_test_data, feature_number):

    cve = []
    te = []
    s = []
    s_test_error = []
    s_cross_validation_error = []
    n = x_train_data.shape[1]
    select_number = 0

    clf = LogisticRegression(random_state = 42).fit(x_train_data, y_train_data)

    for k in range(1, feature_number + 1):
        print('k =', k)
        cross_validation_error = 1

        for i in range(n):
            if i not in s:
                current_accuracy = cross_val_score(clf, x_train_data[:, s+[i]], y_train_data, scoring = 'acc
                current_error = 1 - np.mean(current_accuracy)
                if current_error < cross_validation_error:
                    cross_validation_error = current_error
                    select_number = i

        if select_number not in s:
            s.append(select_number)
            print('s =', s)

        s_cross_validation_error.append(cross_validation_error)
        mse_cross_val_error = np.mean(s_cross_validation_error)
        print('cross validation error =', mse_cross_val_error)

        clf.fit(x_train_data[:,s], y_train_data)
        test_error = 1 - clf.score(x_test_data[:,s], y_test_data)
        s_test_error.append(test_error)
        mse_test_error = np.mean(s_test_error)
        print('test error =', mse_test_error)

        cve.append(mse_cross_val_error)
        te.append(mse_test_error)

        print('\n')
    return s, mse_cross_val_error, mse_test_error, te, cve
```

In [5]:
```python
s, cross_validation_error, test_error, te, cve = sfs_function(X_train, y_train, X_test, y_test, 13)
s, cross_validation_error, test_error, te, cve
```

```
k = 1
s = [2]
cross validation error = 0.24
test error = 0.24271844660194175


k = 2
s = [2, 11]
cross validation error = 0.23249999999999993
test error = 0.27184466019417475


k = 3
s = [2, 11, 12]
cross validation error = 0.2133333333333333
test error = 0.2524271844660194


k = 4
s = [2, 11, 12, 10]
cross validation error = 0.19874999999999998
test error = 0.24514563106796117


k = 5
s = [2, 11, 12, 10, 4]
cross validation error = 0.19
test error = 0.24077669902912618


k = 6
s = [2, 11, 12, 10, 4, 5]
cross validation error = 0.18416666666666667
test error = 0.2378640776699029


k = 7
s = [2, 11, 12, 10, 4, 5, 1]
cross validation error = 0.1814285714285714
test error = 0.23300970873786403


k = 8
s = [2, 11, 12, 10, 4, 5, 1, 7]
cross validation error = 0.17874999999999996
test error = 0.22815533980582525


k = 9
s = [2, 11, 12, 10, 4, 5, 1, 7, 3]
cross validation error = 0.17611111111111105
test error = 0.2243797195253506


k = 10
s = [2, 11, 12, 10, 4, 5, 1, 7, 3, 0]
cross validation error = 0.17399999999999993
test error = 0.22135922330097085


k = 11
s = [2, 11, 12, 10, 4, 5, 1, 7, 3, 0, 6]
cross validation error = 0.17272727272727265
test error = 0.21888790820829654


k = 12
s = [2, 11, 12, 10, 4, 5, 1, 7, 3, 0, 6, 9]
cross validation error = 0.17124999999999993
test error = 0.21521035598705499


k = 13
s = [2, 11, 12, 10, 4, 5, 1, 7, 3, 0, 6, 9, 8]
cross validation error = 0.17076923076923067
test error = 0.2135922330097087
```

Out[5]:
```
([2, 11, 12, 10, 4, 5, 1, 7, 3, 0, 6, 9, 8],
 0.17076923076923067,
 0.2135922330097087,
 [0.24271844660194175,
  0.27184466019417475,
  0.2524271844660194,
  0.24514563106796117,
  0.24077669902912618,
  0.2378640776699029,
  0.23300970873786403,
  0.22815533980582525,
  0.2243797195253506,
  0.22135922330097085,
  0.21888790820829654,
  0.21521035598705499,
  0.2135922330097087],
 [0.24,
  0.23249999999999993,
  0.2133333333333333,
  0.19874999999999998,
  0.19,
  0.18416666666666667,
  0.1814285714285714,
  0.17874999999999996,
  0.17611111111111105,
  0.17399999999999993,
  0.17272727272727265,
  0.17124999999999993,
  0.17076923076923067])
```

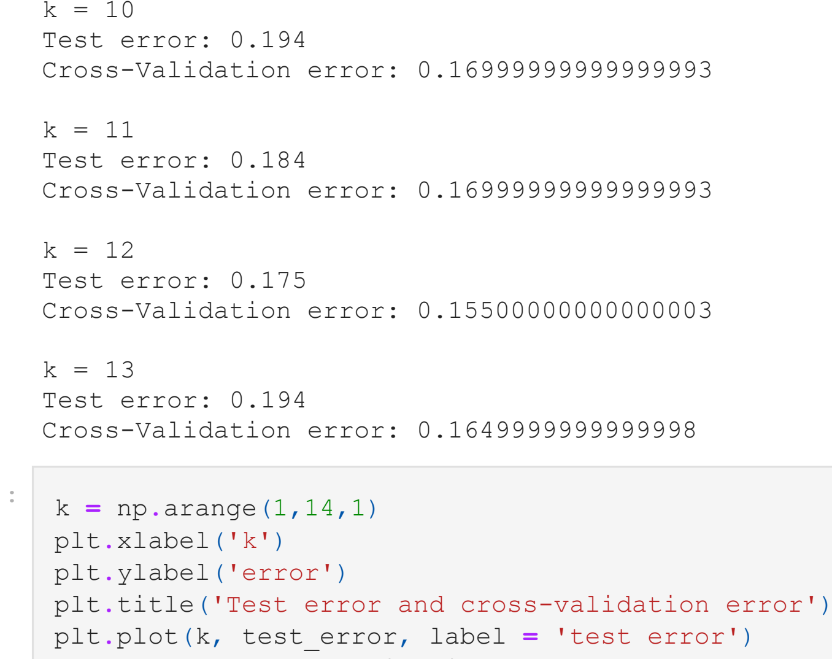
In [6]:
```python
k = np.arange(1,14,1)
plt.xlabel('k')
plt.ylabel('error')
plt.title('Test error and cross-validation error')
plt.plot(k, te, label = 'test error')
plt.plot(k, cve, label = 'cross validation error')
plt.legend()
plt.grid()
plt.show()
```



In [7]:
```python
# SECOND OPTION: call the SequentialFeatureSelector package to compare the results

# Build logistic regression classifier to use in feature selection
clf = LogisticRegression(random_state = 42).fit(X_train, y_train)

test_error = []
cross_validation_error = []

for k in range(1, 14):
    # Build step forward feature selection
    sfs1 = sfs(clf,
               k_features=k,
               forward=True,
               floating=False,
               scoring='accuracy',
               cv=5)
    print('\nk =', k)

    sfs1 = sfs1.fit(X_train, y_train)

    # select features
    feat_cols = list(sfs1.k_feature_idx_)

    # Build model with selected features
    clf = LogisticRegression(random_state=42)
    clf.fit(X_train[:, feat_cols], y_train)

    y_test_pred = clf.predict(X_test[:, feat_cols])

    y_test_error = 1 - acc(y_test, y_test_pred)
    print('Test error: %.3f' % y_test_error)

    test_error.append(y_test_error)

    scores = cross_val_score(clf, X_train[:, feat_cols], y_train, cv=10)

    scores = pd.Series(scores)
    scores.min(), scores.mean(), scores.max()
    error_rate = 1 - scores.mean()
    print('Cross-Validation error:', error_rate)

    cross_validation_error.append(error_rate)
```

```
k = 1
Test error: 0.243
Cross-Validation error: 0.24

k = 2
Test error: 0.301
Cross-Validation error: 0.22499999999999998

k = 3
Test error: 0.214
Cross-Validation error: 0.17499999999999982

k = 4
Test error: 0.204
Cross-Validation error: 0.16500000000000004

k = 5
Test error: 0.204
Cross-Validation error: 0.16500000000000004

k = 6
Test error: 0.204
Cross-Validation error: 0.16500000000000004

k = 7
Test error: 0.204
Cross-Validation error: 0.16999999999999993

k = 8
Test error: 0.204
Cross-Validation error: 0.15999999999999992

k = 9
Test error: 0.204
Cross-Validation error: 0.18000000000000005

k = 10
Test error: 0.194
Cross-Validation error: 0.16999999999999993

k = 11
Test error: 0.184
Cross-Validation error: 0.16999999999999993

k = 12
Test error: 0.175
Cross-Validation error: 0.15500000000000003

k = 13
Test error: 0.194
Cross-Validation error: 0.16499999999999998
```

In [8]:
```python
k = np.arange(1,14,1)
plt.xlabel('k')
plt.ylabel('error')
plt.title('Test error and cross-validation error')
plt.plot(k, test_error, label = 'test error')
plt.plot(k, cross_validation_error, label = 'cross validation error')
plt.legend()
plt.grid()
plt.show()
```



(b) What two features were chosen for k = 2? Plot the decision boundary in this case.

In [9]:
```python
# FIRST OPTION: call the hand written sfs_function

s_2, cross_validation_error_2, test_error_2, te_2, cve_2 = sfs_function(X_train, y_train, X_test, y_test, 2)

feature_1_1 = dataframe.columns[s_2[0]]
feature_2_1 = dataframe.columns[s_2[1]]

print('two features selected:', feature_1_1, feature_2_1)
```

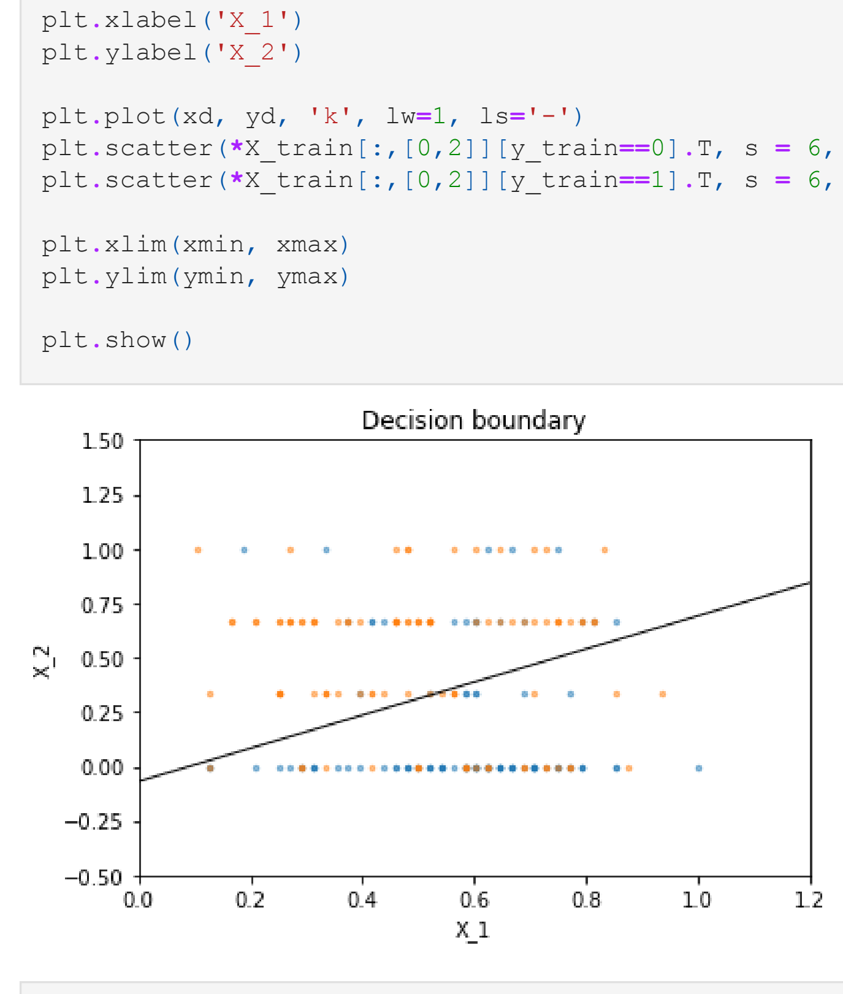
```
k = 1
s = [2]
cross validation error = 0.24
test error = 0.24271844660194175


k = 2
s = [2, 11]
cross validation error = 0.23249999999999993
test error = 0.27184466019417475


two features selected: cp ca
```

In [10]:
```python
# SECOND OPTION: call the SequentialFeatureSelector package to compare the results

# Build step forward feature selection
sfs1 = sfs(clf,
           k_features=2,
           forward=True,
           floating=False,
           scoring='accuracy',
           cv=5)

sfs1 = sfs1.fit(X_train, y_train)
feat_cols = list(sfs1.k_feature_idx_)
#print('Features selected:', feat_cols)

clf = LogisticRegression(random_state=42)
clf.fit(X_train[:, feat_cols], y_train)


feature_1_2 = dataframe.columns[feat_cols[0]]
feature_2_2 = dataframe.columns[feat_cols[1]]

print('two features selected:', feature_1_2, feature_2_2)
```

```
two features selected: cp ca
```

In [11]:
```python
# Retrieve the model parameters.
b = clf.intercept_[0]
w1, w2 = clf.coef_.T

# Calculate the intercept and gradient of the decision boundary.
c = -b/w2
m = -w1/w2
```

In [12]:
```python
# Plot the data and the classification with the decision boundary.
xmin, xmax = 0, 1.2
ymin, ymax = -0.5, 1.5

xd = np.array([xmin, xmax])
yd = m*xd + c

plt.title('Decision boundary')
plt.xlabel('X_1')
plt.ylabel('X_2')

plt.plot(xd, yd, 'k', lw=1, ls='-')
plt.scatter(*X_train[:,[0,2]][y_train==0].T, s = 6, alpha = 0.5)
plt.scatter(*X_train[:,[0,2]][y_train==1].T, s = 6, alpha = 0.5)

plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)

plt.show()
```



In [ ]:

4. Mini-project: Coordinate descent

In [1]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
import pandas as pd
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score as acc
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
import matplotlib
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.model_selection import cross_val_score
from sklearn.metrics import log_loss
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [2]:

```
dataframe = read_csv('heart.csv')
dataframe
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [3]:

```
# Separate features from labels
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
```

(a) A short, high-level description of your coordinate descent method.

In particular, you should give a concise description of how you solve problems (i) and (ii) above. Do you need the function L(·) to be differentiable, or does it work with any loss function?

(i) Which coordinate to choose?

Answer: We will choose the coordinate that minimize the wi by running all coordinates in cyclic order at each iteration.

(ii) How to set the new value of wi?

Answer: For setting the new value of wi, we will use the current value of wi subtract value of the stepsize times gradient.

Do you need the function L(·) to be differentiable, or does it work with any loss function?

Answer: Yes, the function L(·) needs to be differentiable and convex.

Summary of the coordinate descent method

Answer: For the loss function, we will minimize it by minimizing each of the individual dimensions of it in a cyclic fashion, while holding the values of it in the other dimensions fixed.

(b) Convergence.

Under what conditions do you think your method converges to the optimal loss? There's no need to prove anything: just give a few sentences of brief explanation.

Answer: After minimizing the cost function with respect to each coordinate, the cost function has bounded level sets and is in some sense strictly convex.

(c) Experimental results.

Begin by running a standard logistic regression solver (e.g., from scikit-learn) on the training set. It should not be regularized: if the solver forces you to do this, just set the regularization constant suitably to make it irrelevant. Make note of the final loss L∗.

In [4]:

```
# Build logistic regression classifier
clf = LogisticRegression(random_state = 42).fit(X, y)
proba = clf.predict_proba(X)
log_loss_value = log_loss(y, proba)
print('Final loss\n L* =', log_loss_value)
```

```
Final loss
 L* = 0.3534163611333882
```

In [5]:

```
# normalize data
X_normalize = (X - np.mean(X,axis = 0))/(np.max(X, axis = 0) - np.min(X, axis = 0))
intercept = np.ones((X.shape[0], 1))
intercept = intercept.reshape(X.shape[0], 1)
X_norm = np.hstack((X_normalize, intercept))
X_norm.shape
```

Out[5]:

```
(303, 14)
```

Then, implement your coordinate descent method and run it on this data.

In [6]:

```
# predict y
def predict_function(w, x):
    return 1/(1 + (np.exp(-(np.dot(w.T, x.T)))))

# calculate the loss
def loss_function(x, y, y_predict):
    return -(1/x.shape[0]) * np.sum(y * np.log(y_predict) + (1 - y) * np.log(1 - y_predict))

# calculate gradient
def gradient_function(x, y, y_predict):
    return list(np.dot((y_predict-y), x)[0])
```

In [7]:

```
# calculate the loss by using the coordinate descent method
def coordinate_descent(x, y, iteration, etha):
    w = np.zeros((x.shape[1],1))
    index = 0
    loss = []
    n = (x.shape[1] - 1)

    for i in range(iteration):
        y_predict = predict_function(w, x)
        gradient = gradient_function(x, y, y_predict)
        w[index%n] = w[index%n] - etha*gradient[index%n]
        index += 1
        loss.append(loss_function(x, y, y_predict))
    return loss
```

In [8]:

```
loss_coordinate_descent = coordinate_descent(X_norm, y, 1000, 0.01)
print('The minimum loss about the coordinate descent is:\n', min(loss_coordinate_descent))
```

```
The minimum loss about the coordinate descent is:
 0.35857565006240405
```

Finally, compare to a method that chooses coordinates i uniformly at random and then updates wi using your method (we'll call this "random-feature coordinate descent").

In [9]:

```
def random_coordinate_descent(x, y, number_iteration, etha):
    w = np.zeros((x.shape[1],1))
    loss = []
    n = (x.shape[1] - 1)

    for i in range(number_iteration):
        y_predict = predict_function(w, x)
        gradient = gradient_function(x, y, y_predict)
        current_index = np.random.randint(0,n)
        w[current_index] = w[current_index] - etha*gradient[current_index]
        loss.append(loss_function(x, y, y_predict))
    return loss
```

In [10]:

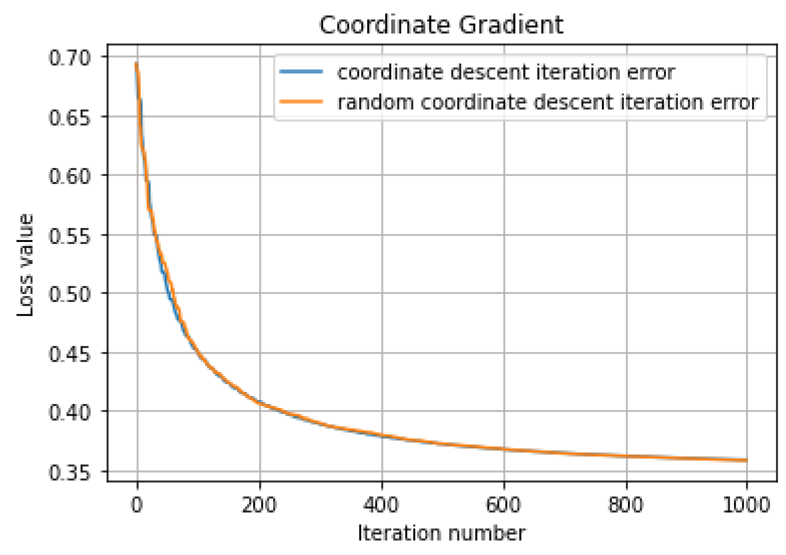
```
loss_random_coordinate_descent = random_coordinate_descent(X_norm, y, 1000, 0.01)
print('The minimum loss about the coordinate descent is:\n', min(loss_random_coordinate_descent))
```

```
The minimum loss about the coordinate descent is:
 0.35817122582262023
```

Produce a clearly-labeled graph that shows how the loss of your algorithm's current iterate— that is, L(wt)— decreases with t; it should asymptote to L∗. On the same graph, show the corresponding curve for random-feature coordinate descent.

In [11]:

```
iteration = np.arange(0,1000)
plt.xlabel('Iteration number')
plt.ylabel('Loss value')
plt.title('Coordinate Gradient')
plt.plot(iteration, loss_coordinate_descent, label = 'coordinate descent iteration error')
plt.plot(iteration, loss_random_coordinate_descent, label = 'random coordinate descent iteration error')
plt.legend()
plt.grid()
plt.show()
```


Coordinate Gradient
coordinate descent iteration error
random coordinate descent iteration error
Loss value
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
200
400
600
800
1000
Iteration number