

In [1]:

```
%load_ext watermark
%watermark -a 'Sebastian Raschka' -u -d -v -p numpy,pandas,matplotlib,scikit-learn
```

UsageError: unrecognized arguments: Raschka'

In [2]:

```
from IPython.display import Image
```

In [3]:

```
%matplotlib inline
```

In [4]:

```
import pandas as pd
from io import StringIO

csv_data = ''' A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''

# If you are using Python 2.7, you need
# to convert the string to unicode:
# csv_data = unicode(csv_data)

df = pd.read_csv(StringIO(csv_data))
df
```

Out[4]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
A    0
B    0
C    1
D    1
dtype: int64
```

In [6]:

```
df.dropna()
```

Out[6]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

In [7]:

```
df.dropna(axis=1)
```

Out[7]:

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

In [8]:

```
# only drop rows where all columns are NaN  
df.dropna(how='all')
```

Out[8]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

In [9]:

```
# drop rows that have not at least 4 non-NaN values  
df.dropna(thresh=4)
```

Out[9]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

In [10]:

```
# only drop rows where NaN appear in specific columns (here: 'C')
df.dropna(subset=['C'])
```

Out[10]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
2	10.0	11.0	12.0	NaN

In [11]:

```
from sklearn.preprocessing import Imputer

imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
imr = imr.fit(df)
imputed_data = imr.transform(df.values)
imputed_data
```

Out[11]:

```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

In [12]:

df.values

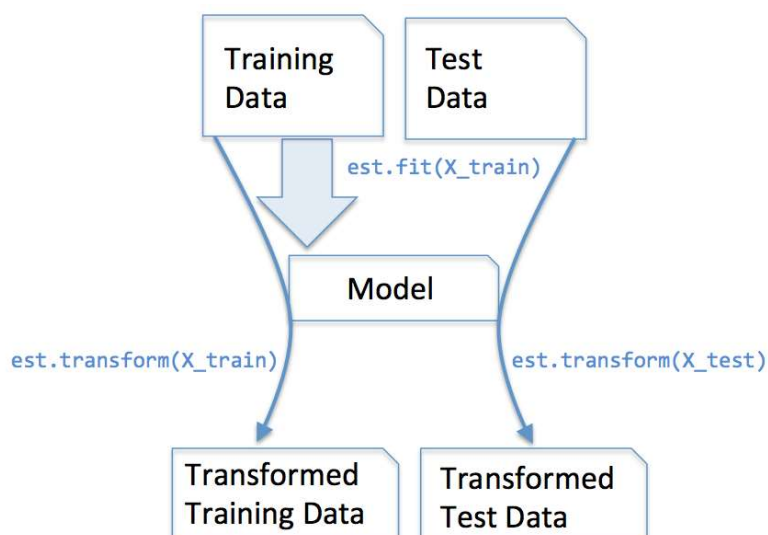
Out[12]:

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., nan,  8.],
       [10., 11., 12., nan]])
```

In [14]:

Image(filename='04_04.png', width=400)

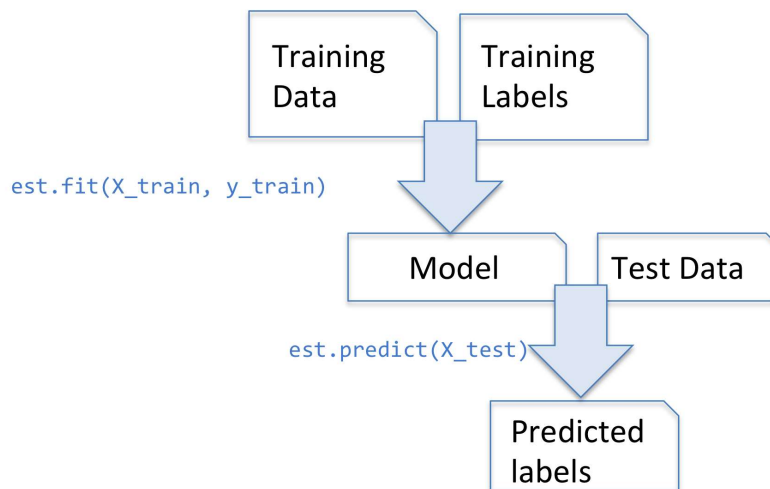
Out[14]:



In [16]:

```
Image(filename='04_05.png', width=400)
```

Out[16]:



In [17]:

```
import pandas as pd

df = pd.DataFrame([['green', 'M', 10.1, 'class1'],
                   ['red', 'L', 13.5, 'class2'],
                   ['blue', 'XL', 15.3, 'class1']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

Out[17]:

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XL	15.3	class1

In [18]:

```
size_mapping = {'XL': 3,
                'L': 2,
                'M': 1}

df['size'] = df['size'].map(size_mapping)
df
```

Out[18]:

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

In [19]:

```
inv_size_mapping = {v: k for k, v in size_mapping.items()}
df['size'].map(inv_size_mapping)
```

Out[19]:

```
0    M
1    L
2   XL
Name: size, dtype: object
```

In [20]:

```
import numpy as np

class_mapping = {label: idx for idx, label in enumerate(np.unique(df['classlabel']))}
class_mapping
```

Out[20]:

```
{'class1': 0, 'class2': 1}
```

In [21]:

```
df['classlabel'] = df['classlabel'].map(class_mapping)
df
```

Out[21]:

	color	size	price	classlabel
0	green	1	10.1	0
1	red	2	13.5	1
2	blue	3	15.3	0

In [22]:

```
inv_class_mapping = {v: k for k, v in class_mapping.items()}
df['classlabel'] = df['classlabel'].map(inv_class_mapping)
df
```

Out[22]:

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

In [23]:

```
from sklearn.preprocessing import LabelEncoder

class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

Out[23]:

```
array([0, 1, 0], dtype=int64)
```

In [24]:

```
class_le.inverse_transform(y)
```

Out[24]:

```
array(['class1', 'class2', 'class1'], dtype=object)
```

In [25]:

```
X = df[['color', 'size', 'price']].values

color_le = LabelEncoder()
X[:, 0] = color_le.fit_transform(X[:, 0])
X
```

Out[25]:

```
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

In [26]:

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(categorical_features=[0])
ohe.fit_transform(X).toarray()
```

Out[26]:

```
array([[ 0. ,  1. ,  0. ,  1. , 10.1],
       [ 0. ,  0. ,  1. ,  2. , 13.5],
       [ 1. ,  0. ,  0. ,  3. , 15.3]])
```

In [27]:

```
pd.get_dummies(df[['price', 'color', 'size']])
```

Out[27]:

	price	size	color_blue	color_green	color_red
0	10.1	1	0	1	0
1	13.5	2	0	0	1
2	15.3	3	1	0	0

In [28]:

```
df_wine = pd.read_csv('https://archive.ics.uci.edu/'
                      'ml/machine-learning-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                  'Alcalinity of ash', 'Magnesium', 'Total phenols',
                  'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                  'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                  'Proline']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

Class labels [1 2 3]

Out[28]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavapher
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39

In [29]:

```
df_wine = pd.read_csv('https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wine/wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
'Alcalinity of ash', 'Magnesium', 'Total phenols',
'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']
df_wine.head()
```

Out[29]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavapher
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39

In [30]:

```
from sklearn.cross_validation import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=0)
```

C:\Users\watanabe shinichi\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
 "This module will be removed in 0.20.", DeprecationWarning)

In [31]:

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

In [32]:

```
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```


In [33]:

```
ex = pd.DataFrame([0, 1, 2, 3, 4, 5])

# standardize
ex[1] = (ex[0] - ex[0].mean()) / ex[0].std(ddof=0)

# Please note that pandas uses ddof=1 (sample standard deviation)
# by default, whereas NumPy's std method and the StandardScaler
# uses ddof=0 (population standard deviation)

# normalize
ex[2] = (ex[0] - ex[0].min()) / (ex[0].max() - ex[0].min())
ex.columns = ['input', 'standardized', 'normalized']
ex
```

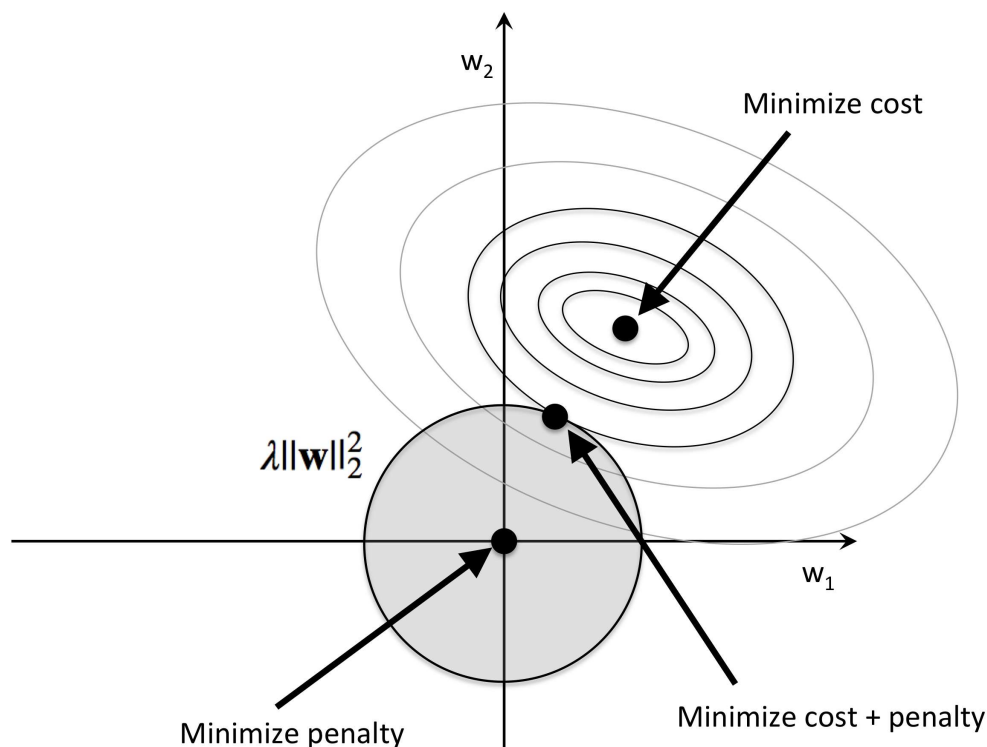
Out[33]:

	input	standardized	normalized
0	0	-1.46385	0.0
1	1	-0.87831	0.2
2	2	-0.29277	0.4
3	3	0.29277	0.6
4	4	0.87831	0.8
5	5	1.46385	1.0

In [36]:

```
Image(filename='04_12.png', width=500)
```

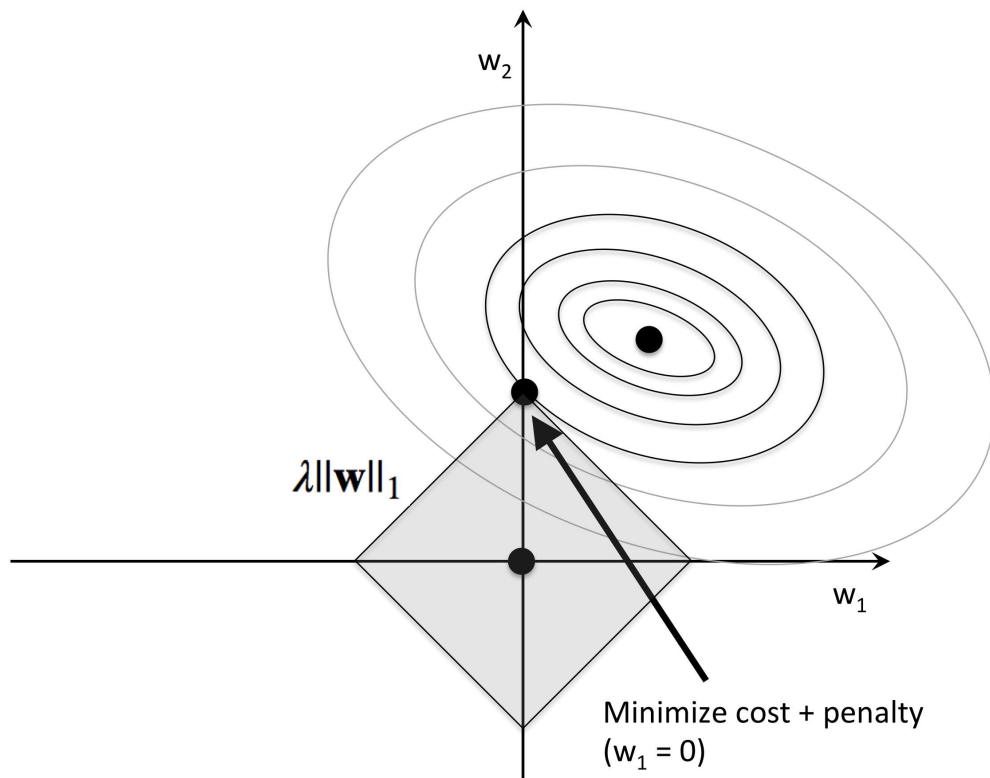
Out[36]:



In [37]:

```
Image(filename='04_13.png', width=500)
```

Out[37]:



In [38]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1', C=0.1)
lr.fit(X_train_std, y_train)
print('Training accuracy:', lr.score(X_train_std, y_train))
print('Test accuracy:', lr.score(X_test_std, y_test))
```

Training accuracy: 0.983870967742

Test accuracy: 0.981481481481

In [39]:

```
lr.intercept_
```

Out[39]:

```
array([-0.38380696, -0.15807113, -0.70045745])
```

In [40]:

```
lr.coef_
```

Out[40]:

```
array([[ 0.28002977,  0.          ,  0.          , -0.02795832,  0.          ,
         0.          ,  0.70996756,  0.          ,  0.          ,  0.          ,
         0.          ,  0.          ,  1.23666297],
       [-0.64400959, -0.06880435, -0.0572097 ,  0.          ,  0.          ,
         0.          ,  0.          ,  0.          ,  0.          , -0.9267139 ,
         0.06017463,  0.          , -0.37105005],
       [ 0.          ,  0.06150922,  0.          ,  0.          ,  0.          ,
         0.          , -0.63544606,  0.          ,  0.          ,  0.49781135,
        -0.35843166, -0.57166647,  0.          ]])
```

In [42]:

```
import matplotlib.pyplot as plt

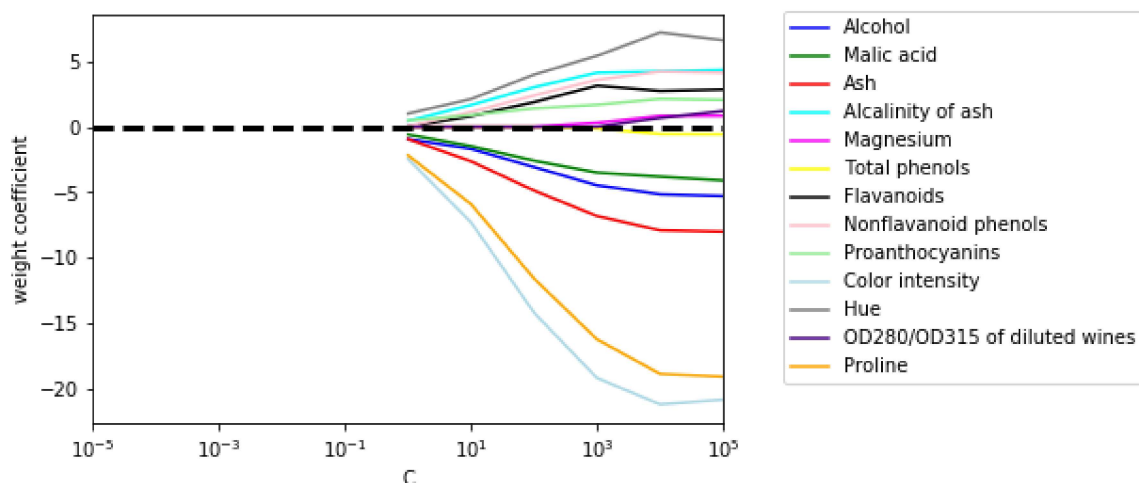
fig = plt.figure()
ax = plt.subplot(111)

colors = ['blue', 'green', 'red', 'cyan',
          'magenta', 'yellow', 'black',
          'pink', 'lightgreen', 'lightblue',
          'gray', 'indigo', 'orange']

weights, params = [], []
for c in np.arange(0, 6):
    lr = LogisticRegression(penalty='l1', C=10**c, random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)

for column, color in zip(range(weights.shape[1]), colors):
    plt.plot(params, weights[:, column],
             label=df_wine.columns[column + 1],
             color=color)
plt.axhline(0, color='black', linestyle='--', linewidth=3)
plt.xlim([10**(-5), 10**5])
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.xscale('log')
plt.legend(loc='upper left')
ax.legend(loc='upper center',
         bbox_to_anchor=(1.38, 1.03),
         ncol=1, fancybox=True)
# plt.savefig('./figures/l1_path.png', dpi=300)
plt.show()
```



In [43]:

```

from sklearn.base import clone
from itertools import combinations
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score

class SBS():
    def __init__(self, estimator, k_features, scoring=accuracy_score,
                  test_size=0.25, random_state=1):
        self.scoring = scoring
        self.estimator = clone(estimator)
        self.k_features = k_features
        self.test_size = test_size
        self.random_state = random_state

    def fit(self, X, y):

        X_train, X_test, y_train, y_test = \
            train_test_split(X, y, test_size=self.test_size,
                            random_state=self.random_state)

        dim = X_train.shape[1]
        self.indices_ = tuple(range(dim))
        self.subsets_ = [self.indices_]
        score = self._calc_score(X_train, y_train,
                                X_test, y_test, self.indices_)
        self.scores_ = [score]

        while dim > self.k_features:
            scores = []
            subsets = []

            for p in combinations(self.indices_, r=dim - 1):
                score = self._calc_score(X_train, y_train,
                                         X_test, y_test, p)
                scores.append(score)
                subsets.append(p)

            best = np.argmax(scores)
            self.indices_ = subsets[best]
            self.subsets_.append(self.indices_)
            dim -= 1

            self.scores_.append(scores[best])
        self.k_score_ = self.scores_[-1]

        return self

    def transform(self, X):
        return X[:, self.indices_]

    def _calc_score(self, X_train, y_train, X_test, y_test, indices):
        self.estimator.fit(X_train[:, indices], y_train)
        y_pred = self.estimator.predict(X_test[:, indices])
        score = self.scoring(y_test, y_pred)
        return score

```

In [44]:

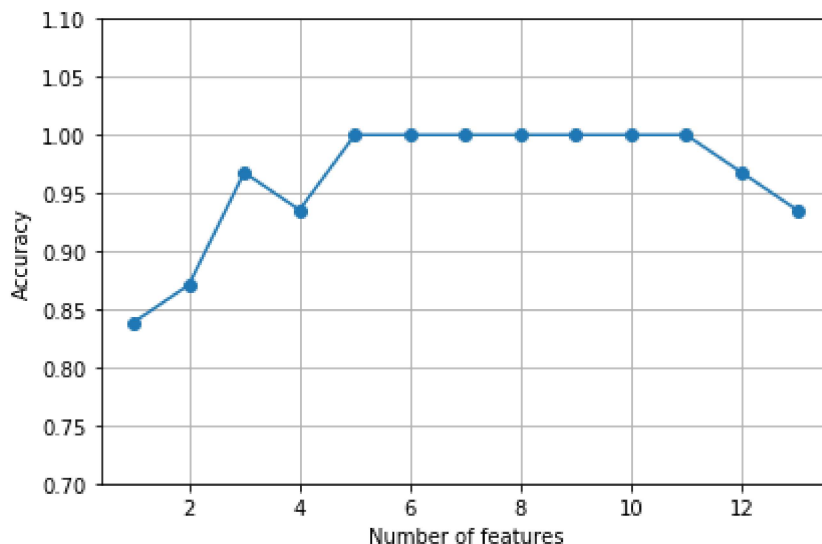
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=2)

# selecting features
sbs = SBS(knn, k_features=1)
sbs.fit(X_train_std, y_train)

# plotting performance of feature subsets
k_feat = [len(k) for k in sbs.subsets_]

plt.plot(k_feat, sbs.scores_, marker='o')
plt.ylim([0.7, 1.1])
plt.ylabel('Accuracy')
plt.xlabel('Number of features')
plt.grid()
plt.tight_layout()
# plt.savefig('./sbs.png', dpi=300)
plt.show()
```



In [45]:

```
k5 = list(sbs.subsets_[8])
print(df_wine.columns[1:][k5])
```

```
Index(['Alcohol', 'Malic acid', 'Alcalinity of ash', 'Hue', 'Proline'], dtype='object')
```

In [46]:

```
knn.fit(X_train_std, y_train)
print('Training accuracy:', knn.score(X_train_std, y_train))
print('Test accuracy:', knn.score(X_test_std, y_test))
```

```
Training accuracy: 0.983870967742
Test accuracy: 0.944444444444
```