K Nearest Neighbors

In [1]:
```
# imports
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import math
import random
from sklearn import preprocessing
from sklearn.utils import column_or_1d
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
%config InlineBackend.figure_format = 'retina'
```

In [2]:
```
# Use the cross-validation function from HW6
def simple_cross_validation(X_train_val, Y_train_val, k, fold):
    """
    A simple cross-validation function for k-NN.

    X_train_val: Features for train and val set.
                 Shape: (num of data points, num of features)
    Y_train_val: Labels for train and val set.
                 Shape: (num of data points,)
    k:           Parameter k for k-NN.
    fold:        The number of folds to do the cross-validation.

    Return the average accuracy on validation set.
    """
    val_acc_list = []
    set_size = int(len(Y_train_val) / fold)
    for i in range(1, fold + 1):
        # select correct fold
        cut =  i * set_size
        # cut out the test set
        X_test_set = X_train_val[cut-set_size:cut,:]
        Y_test_set = Y_train_val[cut-set_size:cut]
        # the rest is the training set
        X_val_set = np.vstack((X_train_val[0:cut-set_size, :], X_train_v
al[cut:,:]))
        Y_val_set = np.vstack((Y_train_val[0:cut-set_size], Y_train_val[
cut:]))
        # Use sklearn's implementation as it is much faster
        classifier  = KNeighborsClassifier(algorithm='auto', n_neighbors
=k)
        # train
        classifier.fit(X_val_set, column_or_1d(Y_val_set))
        #predict on the test set
        predictions = classifier.predict(X_test_set)
        total = 0
        for i in range(0, len(predictions)):
            total += 1 if predictions[i] == Y_test_set[i] else 0
        val_acc = total / len(predictions)
        val_acc_list.append(val_acc)

    return sum(val_acc_list) / len(val_acc_list)
```

In [3]:
```
# And the grid search function as well
def simple_GridSearchCV_fit(X_train_val, Y_train_val, k_list, fold):
    """
    A simple grid search function for k with cross-validation in k-NN.

    X_train_val: Features for train and val set.
                 Shape: (num of data points, num of features)
    Y_train_val: Labels for train and val set.
                 Shape: (num of data points,)
    k_list:      The list of k values to try.
    fold:        The number of folds to do the cross-validation.

    Return the val and train accuracy matrix of cross-validation.
    All combinations of k are included in the array.
    Shape: (len(k_list), )
    """
    val_acc_array   = [0 for i in range(len(k_list))]
    for i in range(len(k_list)):
        val_acc_array[i] = simple_cross_validation(X_train_val, Y_train_
val, k_list[i], fold)
    return val_acc_array
```

In [4]:
```
# heatmap function
def draw_heatmap_knn(acc, acc_desc, k_list):
    plt.figure(figsize = (2,8))
    ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=k_list, xti
cklabels=[])
    ax.collections[0].colorbar.set_label("accuracy")
    ax.set(ylabel='$k$')
    plt.title(acc_desc + ' w.r.t $k$')
    sns.set_style("whitegrid", {'axes.grid' : False})
    plt.show()
```

Adult

In [5]:
```
arr = np.load('adult.npy') # load the data
np.random.shuffle(arr)    # Shuffle the data.
x = arr[:,:-1]    # First column to second last column: Features (numeri
cal values)
y = arr[:,-1:]    # Last column: Labels (0 or 1)
print(arr.shape, x.shape, y.shape)  # Check the shapes.
```

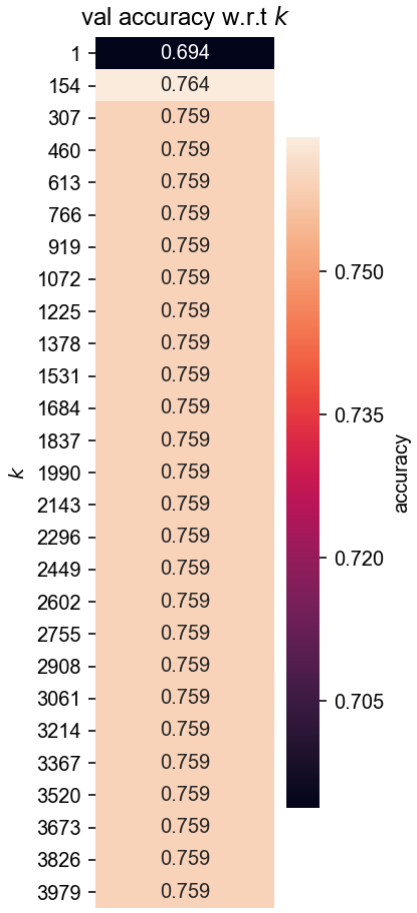
```
(32561, 203) (32561, 202) (32561, 1)
```

In [6]:
```
# Take the first 5000 entries as training data (this will be random as t
he data has been shuffled)
x_train = x[:5000]
y_train = y[:5000]
# the rest is the test set
x_test = x[5000:]
y_test = y[5000:]
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape) # check
 the shapes
```

```
(5000, 202) (5000, 1) (27561, 202) (27561, 1)
```

In [7]:
```
# the paper uses k values from 1 to |training set| with an interval of |
training set| / 26
k_list = [i for i in range(1, 4000, 153)] # 4000 / 26 ~= 153
# Perform grid search
val_acc_array = simple_GridSearchCV_fit(
    x_train,
    y_train,
    k_list,
    5
)
```

In [8]:
```
val_acc_array = np.array(val_acc_array)
draw_heatmap_knn(val_acc_array.reshape(-1,1), 'val accuracy', k_list)
print('Best k: 154')
```




```
Best k: 154
```

In [9]:
```
# Use the best k to calculate the test accuracy.
classifier = KNeighborsClassifier(algorithm='auto', n_neighbors=154)
classifier.fit(x_train, column_or_1d(y_train))
predictions = classifier.predict(x_test)
acc = 0
for i in range(len(predictions)):
    acc += 1 if predictions[i] == y_test[i] else 0
acc /= len(predictions)
print('Test accuracy: {}'.format(acc))
```

```
Test accuracy: 0.7622002104422917
```

Cover Type

In [25]:
```
arr = np.load('covtype.npy') # load the data
np.random.shuffle(arr)    # Shuffle the data.
x = arr[:,:-1]    # First column to second last column: Features (numeri
cal values)
y = arr[:,-1:]    # Last column: Labels (0 or 1)
print(arr.shape, x.shape, y.shape)  # Check the shapes.
```

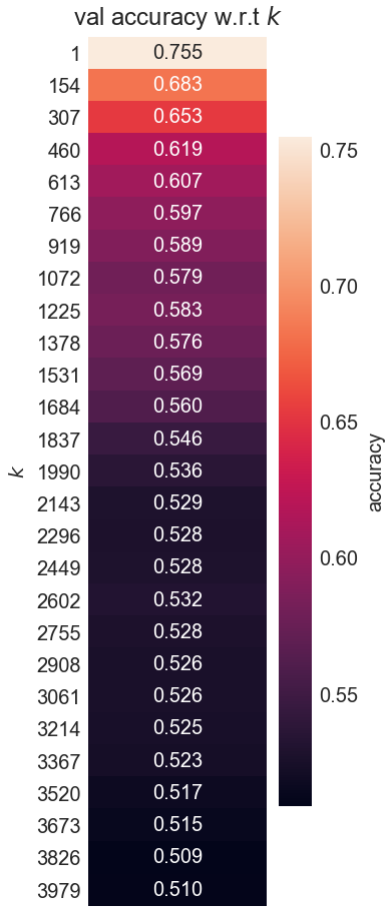
```
(581012, 55) (581012, 54) (581012, 1)
```

In [26]:
```
# Take the first 5000 entries as training data (this will be random as t
he data has been shuffled)
x_train = x[:5000]
y_train = y[:5000]
# the rest is the test set
x_test = x[5000:]
y_test = y[5000:]
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape) # check
 the shapes
```

```
(5000, 54) (5000, 1) (576012, 54) (576012, 1)
```

In [27]:
```
# the paper uses k values from 1 to |training set| with an interval of |
training set| / 26
k_list = [i for i in range(1, 4000, 153)] # |training set| / 26 ~= 153
# Perform grid search
val_acc_array = simple_GridSearchCV_fit(
    x_train,
    y_train,
    k_list,
    5
)
```

In [28]:
```
val_acc_array = np.array(val_acc_array)
draw_heatmap_knn(val_acc_array.reshape(-1,1), 'val accuracy', k_list)
print('Best k: 1')
```




```
Best k: 1
```

In [30]:
```
# Use the best k to calculate the test accuracy.
classifier = KNeighborsClassifier(algorithm='auto', n_neighbors=1)
classifier.fit(x_train, column_or_1d(y_train))
predictions = classifier.predict(x_test)
acc = 0
for i in range(len(predictions)):
    acc += 1 if predictions[i] == y_test[i] else 0
acc /= len(predictions)
print('Test accuracy: {}'.format(acc))
```

```
Test accuracy: 0.7838760303604786
```

Letter P1

In [15]:
```
arr = np.load('letter_p1.npy') # load the data
np.random.shuffle(arr)    # Shuffle the data.
x = arr[:,:-1]    # First column to second last column: Features (numeri
cal values)
y = arr[:,-1:]    # Last column: Labels (0 or 1)
print(arr.shape, x.shape, y.shape)  # Check the shapes.
```

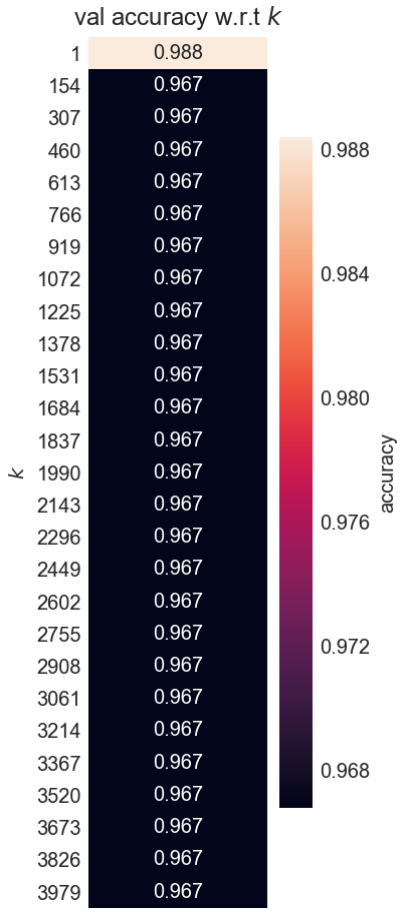
```
(20000, 17) (20000, 16) (20000, 1)
```

In [16]:
```
# Take the first 5000 entries as training data (this will be random as t
he data has been shuffled)
x_train = x[:5000]
y_train = y[:5000]
# the rest is the test set
x_test = x[5000:]
y_test = y[5000:]
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape) # check
 the shapes
```

```
(5000, 16) (5000, 1) (15000, 16) (15000, 1)
```

In [17]:
```
# the paper uses k values from 1 to |training set| with an interval of |
training set| / 26
k_list = [i for i in range(1, 4000, 153)] # |training set| / 26 ~= 153
# Perform grid search
val_acc_array = simple_GridSearchCV_fit(
    x_train,
    y_train,
    k_list,
    5
)
```

In [18]:
```
val_acc_array = np.array(val_acc_array)
draw_heatmap_knn(val_acc_array.reshape(-1,1), 'val accuracy', k_list)
print('Best k: 1')
```




```
Best k: 1
```

In [19]:
```
# Use the best k to calculate the test accuracy.
classifier = KNeighborsClassifier(algorithm='auto', n_neighbors=1)
classifier.fit(x_train, column_or_1d(y_train))
predictions = classifier.predict(x_test)
acc = 0
for i in range(len(predictions)):
    acc += 1 if predictions[i] == y_test[i] else 0
acc /= len(predictions)
print('Test accuracy: {}'.format(acc))
```

```
Test accuracy: 0.9908
```

Letter P2

In [20]:
```
arr = np.load('letter_p2.npy') # load the data
np.random.shuffle(arr)    # Shuffle the data.
x = arr[:,:-1]    # First column to second last column: Features (numeri
cal values)
y = arr[:,-1:]    # Last column: Labels (0 or 1)
print(arr.shape, x.shape, y.shape)  # Check the shapes.
```

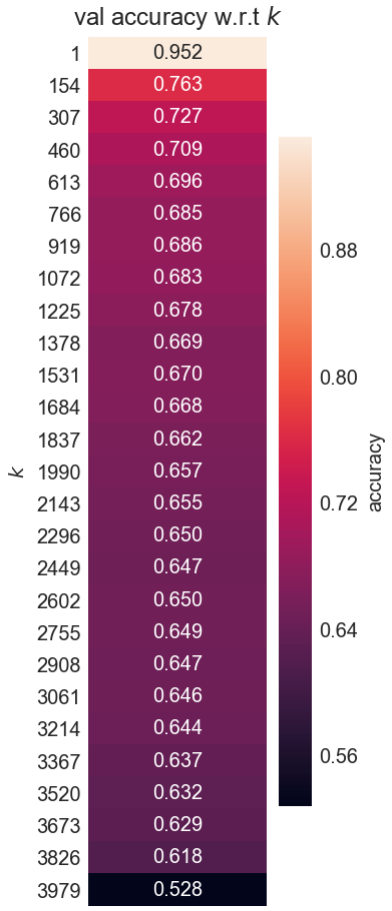
```
(20000, 17) (20000, 16) (20000, 1)
```

In [21]:
```
# Take the first 5000 entries as training data (this will be random as t
he data has been shuffled)
x_train = x[:5000]
y_train = y[:5000]
# the rest is the test set
x_test = x[5000:]
y_test = y[5000:]
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape) # check
 the shapes
```

```
(5000, 16) (5000, 1) (15000, 16) (15000, 1)
```

In [22]:
```
# the paper uses k values from 1 to |training set| with an interval of |
training set| / 26
k_list = [i for i in range(1, 4000, 153)] # |training set| / 26 ~= 153
# Perform grid search
val_acc_array = simple_GridSearchCV_fit(
    x_train,
    y_train,
    k_list,
    5
)
```

In [23]:
```
val_acc_array = np.array(val_acc_array)
draw_heatmap_knn(val_acc_array.reshape(-1,1), 'val accuracy', k_list)
print('Best k: 1')
```




```
Best k: 1
```

In [24]:
```
# Use the best k to calculate the test accuracy.
classifier = KNeighborsClassifier(algorithm='auto', n_neighbors=1)
classifier.fit(x_train, column_or_1d(y_train))
predictions = classifier.predict(x_test)
acc = 0
for i in range(len(predictions)):
    acc += 1 if predictions[i] == y_test[i] else 0
acc /= len(predictions)
print('Test accuracy: {}'.format(acc))
```

```
Test accuracy: 0.9564
```