Sucheta Naik, Statistics and Machine Learning

Introduction:

Iris dataset includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

In this python code, we first import iris dataset, nearest neighbor classifier(knn) and cross validation function from the scikit package.

In [28]:
```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection  import cross_val_score
import matplotlib.pyplot as plt
%matplotlib inline
```

Load the dataset:

In this snippet, we load the dataset and create 2 objects. Create X (features) and y (response)

In [29]:
```
iris=load_iris()

x=iris.data
y=iris.target
```

10-fold cross-validation with K=5 for KNN (the n_neighbors parameter)

In [30]:
```
knn=KNeighborsClassifier(n_neighbors=5)
scores=cross_val_score(knn,x,y,cv=10,scoring='accuracy')
print (scores)
```

```
[ 1.          0.93333333  1.          1.          0.86666667  0.93
333333
  0.93333333  1.          1.          1.        ]
```

In this we use average accuracy as an estimate of out-of-sample accuracy

In [5]:
```
print (scores.mean())
```

```
0.966666666667
```

Search for an optimal value of K for KNN

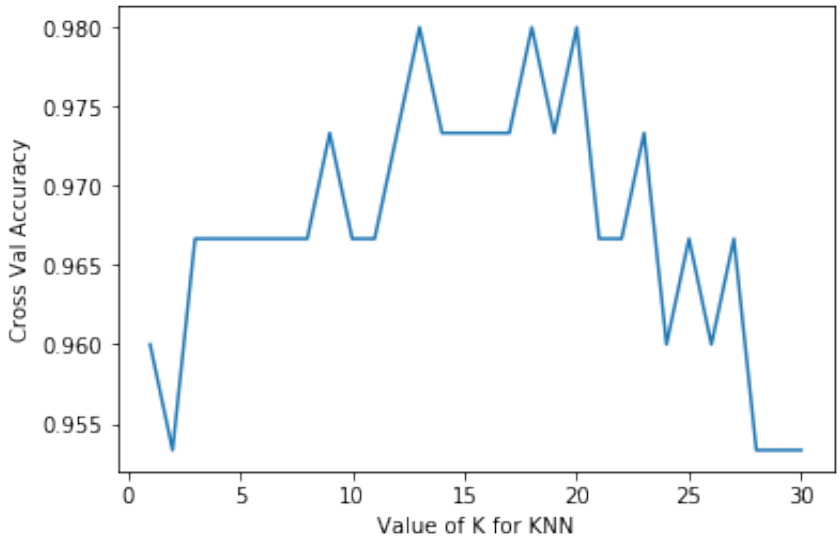
In [6]:
```
k_range=range(1,31)
k_score=[]
for k in k_range:
    knn=KNeighborsClassifier(n_neighbors=k)
    scores=cross_val_score(knn,x,y,cv=10,scoring='accuracy')
    k_score.append(scores.mean())
print (k_score)
```

```
[0.95999999999999996, 0.95333333333333337, 0.96666666666666656, 0.
96666666666666656, 0.96666666666666679, 0.96666666666666679, 0.966
66666666666679, 0.96666666666666679, 0.97333333333333338, 0.966666
66666666679, 0.96666666666666679, 0.97333333333333338, 0.980000000
00000009, 0.97333333333333338, 0.97333333333333338, 0.973333333333
33338, 0.97333333333333338, 0.98000000000000009, 0.973333333333333
38, 0.98000000000000009, 0.96666666666666656, 0.96666666666666656,
0.97333333333333338, 0.95999999999999996, 0.96666666666666656, 0.9
5999999999999996, 0.96666666666666656, 0.95333333333333337, 0.9533
3333333333337, 0.95333333333333337]
```

Plot the value of K for KNN (x-axis) versus the cross-validated accuracy (y-axis)

In [7]:
```
plt.plot(k_range,k_score)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross Validated Accuracy')
```

Out[7]:
```
Text(0,0.5,'Cross Val Accuracy')
```



We import Gridsearch Cross Validation function from scikit package

In [32]:
```
from sklearn.model_selection  import GridSearchCV
```

Search for an optimal value of K for KNN

In [33]:
```
k_range=range(1,31)
print (k_range)
param_grid=dict(n_neighbors=k_range)
print (param_grid)
```

```
range(1, 31)
{'n_neighbors': range(1, 31)}
```

In [34]:
```
grid=GridSearchCV(knn,param_grid,cv=10,scoring='accuracy')
grid.fit(x,y)
```

Out[34]:
```
GridSearchCV(cv=10, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=
30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': range(1, 31)}, pre_dispatch='2*n
_jobs',
       refit=True, return_train_score='warn', scoring='accuracy',
       verbose=0)
```

View complete results

In [14]:
```
grid.grid_scores_
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_se
arch.py:761: DeprecationWarning: The grid_scores_ attribute was de
precated in version 0.18 in favor of the more elaborate cv_results
_ attribute. The grid_scores_ attribute will not be available from
0.20
  DeprecationWarning)
```

```
Out[14]: [mean: 0.96000, std: 0.05333, params: {'n_neighbors': 1},
          mean: 0.95333, std: 0.05207, params: {'n_neighbors': 2},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 3},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 4},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 5},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 6},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 7},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 8},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 9},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 10},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 11},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 12},
          mean: 0.98000, std: 0.03055, params: {'n_neighbors': 13},
          mean: 0.97333, std: 0.04422, params: {'n_neighbors': 14},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 15},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 16},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 17},
          mean: 0.98000, std: 0.03055, params: {'n_neighbors': 18},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 19},
          mean: 0.98000, std: 0.03055, params: {'n_neighbors': 20},
          mean: 0.96667, std: 0.03333, params: {'n_neighbors': 21},
          mean: 0.96667, std: 0.03333, params: {'n_neighbors': 22},
          mean: 0.97333, std: 0.03266, params: {'n_neighbors': 23},
          mean: 0.96000, std: 0.04422, params: {'n_neighbors': 24},
          mean: 0.96667, std: 0.03333, params: {'n_neighbors': 25},
          mean: 0.96000, std: 0.04422, params: {'n_neighbors': 26},
          mean: 0.96667, std: 0.04472, params: {'n_neighbors': 27},
          mean: 0.95333, std: 0.04269, params: {'n_neighbors': 28},
          mean: 0.95333, std: 0.04269, params: {'n_neighbors': 29},
          mean: 0.95333, std: 0.04269, params: {'n_neighbors': 30}]
```

Examine results from first tuple

In [15]:
```
print (grid.grid_scores_[0].parameters)
print (grid.grid_scores_[0].cv_validation_scores)
print (grid.grid_scores_[0].mean_validation_score)
```

```
{'n_neighbors': 1}
[ 1.          0.93333333  1.          0.93333333  0.86666667  1.
  0.86666667  1.          1.          1.        ]
0.96
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_se
arch.py:761: DeprecationWarning: The grid_scores_ attribute was de
precated in version 0.18 in favor of the more elaborate cv_results
_ attribute. The grid_scores_ attribute will not be available from
0.20
  DeprecationWarning)
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_se
arch.py:761: DeprecationWarning: The grid_scores_ attribute was de
precated in version 0.18 in favor of the more elaborate cv_results
_ attribute. The grid_scores_ attribute will not be available from
0.20
  DeprecationWarning)
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_se
arch.py:761: DeprecationWarning: The grid_scores_ attribute was de
precated in version 0.18 in favor of the more elaborate cv_results
_ attribute. The grid_scores_ attribute will not be available from
0.20
  DeprecationWarning)
```

check the results of the grid mean search

In [16]:
```
grid_mean_scores=[result.mean_validation_score for result in grid.g
rid_scores_]
print (grid_mean_scores)
```

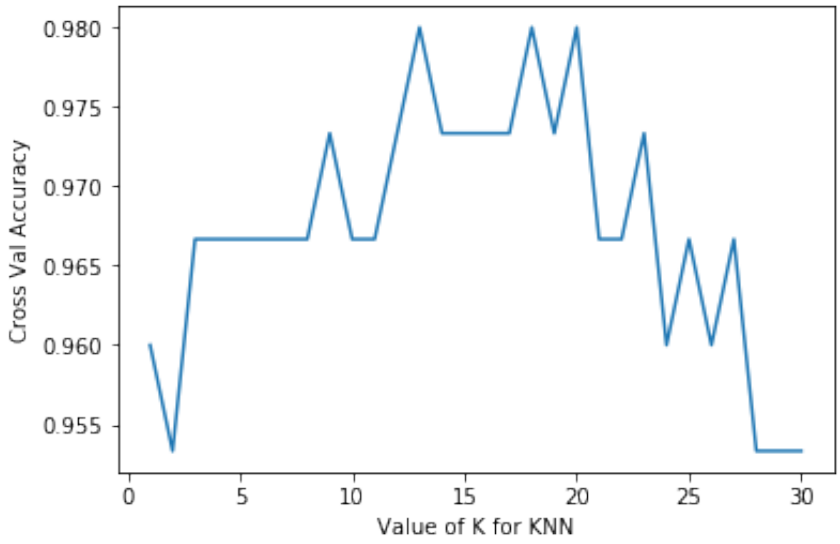
```
[0.95999999999999996, 0.95333333333333337, 0.96666666666666667, 0.
96666666666666667, 0.96666666666666667, 0.96666666666666667, 0.966
66666666666667, 0.96666666666666667, 0.97333333333333338, 0.966666
66666666667, 0.96666666666666667, 0.97333333333333338, 0.979999999
99999998, 0.97333333333333338, 0.97333333333333338, 0.973333333333
33338, 0.97333333333333338, 0.97999999999999998, 0.973333333333333
38, 0.97999999999999998, 0.96666666666666667, 0.96666666666666667,
0.97333333333333338, 0.95999999999999996, 0.96666666666666667, 0.9
5999999999999996, 0.96666666666666667, 0.95333333333333337, 0.9533
3333333333337, 0.95333333333333337]
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_se
arch.py:761: DeprecationWarning: The grid_scores_ attribute was de
precated in version 0.18 in favor of the more elaborate cv_results
_ attribute. The grid_scores_ attribute will not be available from
0.20
  DeprecationWarning)
```

plot the value of K for KNN (x-axis) versus the cross-validated accuracy (y-axis)

In [17]:
```
plt.plot(k_range,grid_mean_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross Val Accuracy')
```

Out[17]:
```
Text(0,0.5,'Cross Val Accuracy')
```



Print the best score,best parameter and best estimator

In [18]:
```
print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
```

```
0.98
{'n_neighbors': 13}
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minko
wski',
           metric_params=None, n_jobs=1, n_neighbors=13, p=2,
           weights='uniform')
```

In [19]:
```
weight_options=['uniform','distance']
```

Search multiple parameters simultaneously

In [36]:
```
param_grid=dict(n_neighbors=k_range,weights=weight_options)
print (param_grid)
grid=GridSearchCV(knn,param_grid,cv=10,scoring='accuracy')
grid.fit(x,y)
grid.grid_scores_
```

```
{'n_neighbors': range(1, 31), 'weights': ['uniform', 'distance']}
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py:761: DeprecationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of the more elaborate cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[36]:
```
[mean: 0.96000, std: 0.05333, params: {'n_neighbors': 1, 'weights': 'uniform'},
 mean: 0.96000, std: 0.05333, params: {'n_neighbors': 1, 'weights': 'distance'},
 mean: 0.95333, std: 0.05207, params: {'n_neighbors': 2, 'weights': 'uniform'},
 mean: 0.96000, std: 0.05333, params: {'n_neighbors': 2, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 3, 'weights': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 3, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 4, 'weights': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 4, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 5, 'weights': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 5, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 6, 'weights': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 6, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 7, 'weights': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 7, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 8, 'weights': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 8, 'weights': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 9, 'weights': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 9, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 10, 'weights': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 10, 'weights': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 11, 'weights': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 11, 'weights': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 12, 'weights
```

```
': 'uniform'},
 mean: 0.97333, std: 0.04422, params: {'n_neighbors': 12, 'weights
': 'distance'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 13, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 13, 'weights
': 'distance'},
 mean: 0.97333, std: 0.04422, params: {'n_neighbors': 14, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 14, 'weights
': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 15, 'weights
': 'uniform'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 15, 'weights
': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 16, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 16, 'weights
': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 17, 'weights
': 'uniform'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 17, 'weights
': 'distance'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 18, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 18, 'weights
': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 19, 'weights
': 'uniform'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 19, 'weights
': 'distance'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 20, 'weights
': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 20, 'weights
': 'distance'},
 mean: 0.96667, std: 0.03333, params: {'n_neighbors': 21, 'weights
': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 21, 'weights
': 'distance'},
 mean: 0.96667, std: 0.03333, params: {'n_neighbors': 22, 'weights
': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 22, 'weights
': 'distance'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 23, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 23, 'weights
': 'distance'},
 mean: 0.96000, std: 0.04422, params: {'n_neighbors': 24, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 24, 'weights
': 'distance'},
 mean: 0.96667, std: 0.03333, params: {'n_neighbors': 25, 'weights
': 'uniform'},
```

```
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 25, 'weights
': 'distance'},
 mean: 0.96000, std: 0.04422, params: {'n_neighbors': 26, 'weights
': 'uniform'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 26, 'weights
': 'distance'},
 mean: 0.96667, std: 0.04472, params: {'n_neighbors': 27, 'weights
': 'uniform'},
 mean: 0.98000, std: 0.03055, params: {'n_neighbors': 27, 'weights
': 'distance'},
 mean: 0.95333, std: 0.04269, params: {'n_neighbors': 28, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 28, 'weights
': 'distance'},
 mean: 0.95333, std: 0.04269, params: {'n_neighbors': 29, 'weights
': 'uniform'},
 mean: 0.97333, std: 0.03266, params: {'n_neighbors': 29, 'weights
': 'distance'},
 mean: 0.95333, std: 0.04269, params: {'n_neighbors': 30, 'weights
': 'uniform'},
 mean: 0.96667, std: 0.03333, params: {'n_neighbors': 30, 'weights
': 'distance'}]
```

In [ ]:
```
Print best score and best parameter
```

In [24]:
```
print (grid.best_score_)
print (grid.best_params_)
```

```
0.98
{'n_neighbors': 13, 'weights': 'uniform'}
```

Predict the response

In [25]:
```
knn=KNeighborsClassifier(n_neighbors=13,weights='uniform')
knn.fit(x,y)
new=([3,5,4,2],[5,4,3,2])
```

In [26]:
```
knn.predict(new)
```

Out[26]:
```
array([1, 1])
```

Accuracy is 98% and there is 2% misclassification. A total of 1 out of 51, or 2 percent of Species were incorrectly classified by the kNN classifier.Therefore 98% accuracy is achieved and is showing by the model with mean: 0.98000, std: 0.03055, params: {'n_neighbors': 13, 'weights': 'uniform'}