In [2]:
```
# 导入KNN
%matplotlib inline
import scipy.io as spio
import numpy as np
import pandas as pd
import sklearn
import os
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

In [3]:
```
batch_size = 100
path ='./'
```

In [4]:
```
mnist = datasets.load_digits()
```

In [5]:
```
# Training and testing split,
# 75% for training and 25% for testing
(trainData, testData, trainLabels, testLabels) = train_test_split(np.array(mnist.dat

# take 10% of the training data and use that for validation
(trainData, valData, trainLabels, valLabels) = train_test_split(trainData, trainLabe
```

In [6]:
```
print("training data points: {}".format(len(trainLabels)))
print("validation data points: {}".format(len(valLabels)))
print("testing data points: {}".format(len(testLabels)))
```

```
training data points: 1212
validation data points: 135
testing data points: 450
```

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
kVals = range(1, 30, 2)
accuracies = []

# loop over kVals
for k in kVals:
    # train the classifier with the current value of `k`
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(trainData, trainLabels)

    # evaluate the model and print the accuracies list
    score = model.score(valData, valLabels)
    print("k=%d, accuracy=%.2f%%" % (k, score * 100))
    accuracies.append(score)
```

In [ ]:
```
from sklearn.metrics import classification_report
# largest accuracy
# np.argmax returns the indices of the maximum values along an axis
i = np.argmax(accuracies)
print("k=%d achieved highest accuracy of %.2f%% on validation data" % (kVals[i],
    accuracies[i] * 100))


# Now that I know the best value of k, re-train the classifier
model = KNeighborsClassifier(n_neighbors=kVals[i])
model.fit(trainData, trainLabels)

# Predict labels for the test set
predictions = model.predict(testData)
```

```
# Evaluate performance of model for each of the digits
print("EVALUATION ON TESTING DATA")
print(classification_report(testLabels, predictions))
```

In [ ]:
```
from yellowbrick.classifier import ClassificationReport
```

In [ ]:
```
model = KNeighborsClassifier(n_neighbors=kVals[i])
model.fit(trainData, trainLabels)
classes = range(10)
visualizer = ClassificationReport(model, classes=classes, support=True)

visualizer.fit(trainData, trainLabels)  # Fit the visualizer and the model
visualizer.score(testData, testLabels)  # Evaluate the model on the test data

g = visualizer.poof()             # Draw/show/poof the data
```

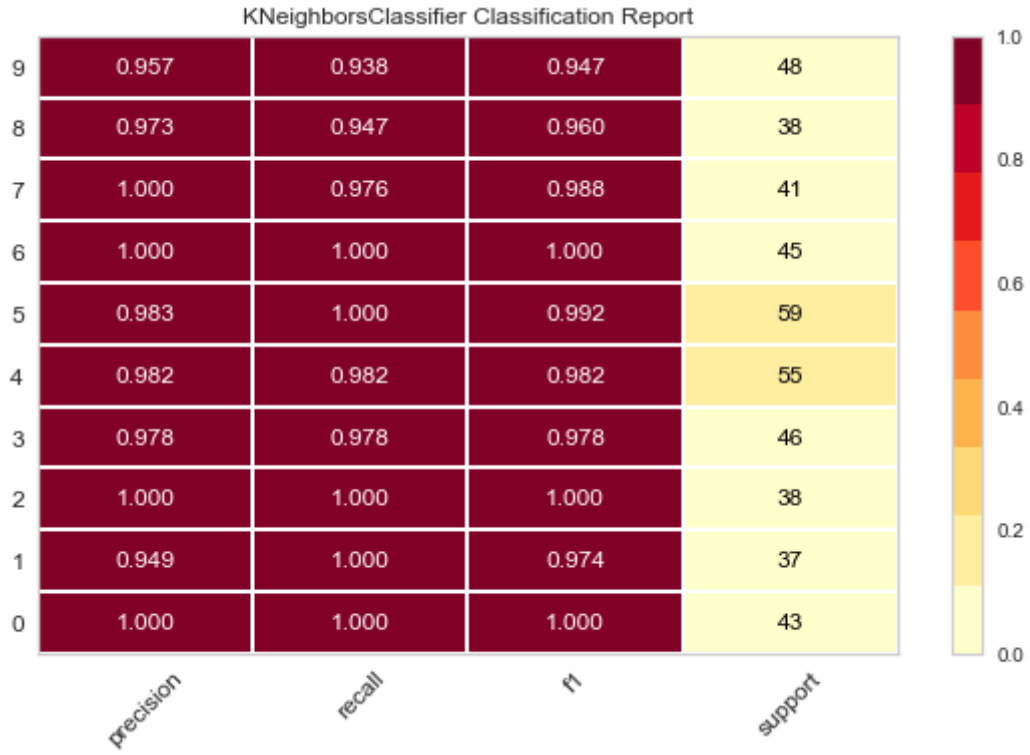
In [ ]:
```
# matplotlib draw graph
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.figure()
plt.plot(kVals, accuracies, label="accuracies")
plt.title("k-NN: k vs. Accuracy")
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.show()
```

In [10]:
```
from yellowbrick.classifier import ClassificationReport
```

In [11]:
```
model = KNeighborsClassifier(n_neighbors=kVals[i])
model.fit(trainData, trainLabels)
classes = range(10)
visualizer = ClassificationReport(model, classes=classes, support=True)

visualizer.fit(trainData, trainLabels)  # Fit the visualizer and the model
visualizer.score(testData, testLabels)  # Evaluate the model on the test data

g = visualizer.poof()             # Draw/show/poof the data
```



In [12]:
```
# matplotlib draw graph
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.figure()
plt.plot(kVals, accuracies, label="accuracies")
plt.title("k-NN: k vs. Accuracy")
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.show()
```