Lab 4 k-Nearest Neighbors

Author: Nigel Nelson

Introduction:

- This lab acts as an introduction to one of the most fundamental non-linear models, the k-nearest neighbors algorithm. KNN is a non-parametric model that is capable of regression or classification. This lab seeks to increase the understanding of KNN by first implementing the model in the "knn" python file that is used in this notebook. Following this, experiments are conducted on several different data sets to explore the visual representation of the nonlinear decision boundaries that KNN is capable of creating, and also augmenting those decision boundaries to see how the boundaries are created in relation to the utilized training data. Lastly, this notebook runs an experiment to explore the impact that k value has on the resulting accuracy of KNN's predictions. This is done utilizing a stratified k-fold data split, so that the effects of incrementally increasing k value can be seen in relation to the average accuracy of the KNN predictions.

Summary:

- This lab acted as an introduction and exploration into the k-nearest neighbors algorithm. Through the exercises and experiments conducted in this lab, many realizations were made. The first glaringly obvious discovery is that KNN is capable of creating nonlinear decision boundaries to capture the nonlinear relationships between data points. However, it was also discovered through experimentation that the value of k can have a profound impact on the efficacy of KNN. By varying the k-value on a given data set, and recording the average accuracy for each series of predictions, it was discovered how important an optimized k-value is. Through these experiments it was seen that a KNN with too small of a k-value can overfit the training data, and a KNN with too large of a k-value can underfit and over generalize the feature space. Specifically, it was found that for the "sweep" data set, and for k-values in steps of 10 between 1 and 200, that a k-value of **20** provided the highest average accuracy with a value of **~0.8720**.

Reflection Questions

KNN Implementation (Problem 1)

1. Estimate the run time complexity in big-o notation of training a KNN model. Justify your answer.
   - The big-o notation should be O(1), or a constant time operation. The reason for this is that the "training" process for KNN simply involves assigned class variables to a provided feature matrix, and an associated label matrix. So, unless a matrix cloning technique is

used that uses looping, the training should simply be an example of variable assignment, which has a big-o notation of O(1).

2. Estimate the run time complexity in big-o notation of predicting the output value for a query point using a training set of N points, with p features, and k neighbors with a KNN model. Justify your answer.
    - The big-o notation for predicting output for a KNN model should be O(Np + N*Log(N) + k), which simplifies to O(N*Log(N)). The reason for this is that for each training point N, p features must be used to calculate the distance between the training point and the target point. Next each of the N distances that were just calculated must be iterated through to sort distances from shortest, to longest distance. The implementation used in this notebook uses numpy.argsort(), which by default uses the 'quicksort' method, which has a big-o run-time of N*Log(N). Finally, to calculate the mode or average, k neighbors must be iterated through to find the average/mode. However, big-o simplifies to whichever is the largest term, which in this case is N*Log(N), resulting in a final big-o of O(N*Log(N)).
3. What do you think the potential downsides of the k nearest neighbors algorithm are? Why do you think it might not be used as widely as other methods?
    - One of the largest downsides must be in relation to the previous answer's big-o analysis. Many deep learning models require higher compute times to train, but after this phase, predicting is a fairly quick process. However, for KNN, each new prediction requires iterating through the entire collection of training points to calculate the distances to the target point. For applications that require frequent predictions, this is a glaring downside. Due to this lack of true training, KNN also requires high memory requirements. As mentioned earlier, many algorithms use the training set once to train, and don't require much memory for the resulting model. However, KNN needs its training data at all times to make predictions, and in problem spaces that require a large amount of data to make accurate predictions, this results in a large amount of accessible storage needed at all times. In addition, KNN suffers from the curse of dimensionality. This results in KNN's predictions losing accuracy as the number of features grows. This is because as the number of features grows, the feature space grows exponentially in size and makes neighboring points further and further away.

Decision Boundaries (Problem 2)

1. For each of the three data sets, do you think a linear decision boundary could be used to accurately classify the data points?
    - For the three datasets, I don't believe that a linear boundary could be used to *accurately* classify the data points. For the moons data set, about a third of each class's data points swirl together in the middle of the graph. This would result in a linear decision boundary cutting down the middle of that swirl and resulting in an inaccurate prediction for most data points that falls within that swirl. The data set that a linear decision boundary would be the least accurate at classifying would be the circles data set. This is because in feature space, class 1 data points form a circle within class 0 data points. If a linear decision boundary was used here, there would be no efficient way to linearly divide the data points due to the circular grouping of classes. The rocky ridge data set provides the most promising use case for a linear decision boundary. However, the linear separation between the classes is still by no means perfect, as such, KNN is more accurate in its separation of classes for this dataset as well.
2. What do we mean by a "non-linear" decision boundary? Give an example of a non-linear function that could be used as a decision boundary for one of the data sets.

- A non-linear decision boundary is a line whose slope does not change over the expanse of the feature space. A non-linear function that could be used as a decision boundary would be the (x-a)^2 + (y-b)^2 = r^2, which is the equation of a circle and could be used with some degree of accuracy to classify the circles data set.

3. What are the advantages of non-linear decision boundaries over linear decision boundaries? What are the disadvantages?
    - The advantage of non-linear decision boundaries is that they are able to capture non-linear relationships between features in a data set, which is an entire division of data that linear decision boundaries fail to accurately classify on their own. A downside of non-linear decision boundaries is that for most cases, they are much less intuitive to use. Most people who have taken a high school math course can grasp how to manipulate the slope and y intercept of a straight line, making adjustment of linear decision boundaries very intuitive. However, with non-linear decision boundaries the underlying mathematical functions at work are not usually obvious at first glance, making adjustments and interpretation comparatively more difficult than linear decision boundaries. In addition, non-linear decision boundaries can be computationally more expensive due to their more complicated nature, which could come a come at a cost when target values are being predicted, or when the model is attempted to be visualized using plots.

Choosing an Optimal Value for k (Problem 3)

1. What value of k gave the highest accuracy?
    - The k-value that gave the highest accuracy, was a k-value of 20, whose average accuracy was 0.8720. A k-value of 20 was just slightly more accurate than a k-value of 40, whose average accuracy was 0.8715. While these numbers will change from experiment to experiment, these two values were generally the highest performing k-values.
2. For large values of k, what happened to the accuracy? Why do you think this is?
    - For larger values of k, the average accuracy plateaued around an average accuracy of 0.784, which happened at a k value of 100. The likely reason for this is that there were an unequal number of class examples. This means that once the k value reached a large enough size, it would classify any data point as the class who had more examples, which was correct for the testing data about 78% of the time.
3. Let's say that we ask you to use the following experimental setup to find a value of k that maximizes the accuracy of the model's predictions: split the data set into training and testing sets, train a model for each value of k using the training set, predictions the classes of the testing points, and evaluate the accuracy of the predictions. Could this approach give you misleading results? If so, why?
    - This experimental setup could lead to misleading results. The reason for this that the data is only being split once. By splitting the data only a single time, the distribution of points is being left up to chance. A certain split on a data set may result in a k value of 3 being ideal, whereas a different split on the same data set could result in a k value of 8 being ideal. By only splitting a single time the validity of the results is somewhat left to chance, whereas if the average accuracy across multiple splits is used, general trends can be seen in the effect that k has on the accuracy of the resulting model.
4. It is considered a "best practice" to use cross-fold validation when optimizing parameters. Why do you think that is
    - I believe that it is a best practice to use cross-fold validation when optimizing parameters. This is due to similar reasons as the answered above. By using cross-fold validation,

chance is taken out of the optimizing equation because there is no longer a reliance on the fact that a given fold is an accurate generalization of the underlying data point patterns. Cross-fold validation allows one to test parameters on a variety of different folds, so that the general trend of a parameter's effect can be seen, instead of a single snapshot of how a parameter affects a random data set split.

Imports:

In [1]:
```
from test_knn import *
from knn import *
import matplotlib.pyplot as plt
from scipy import spatial
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedKFold
```

Verifying KNN implementation passes provided unit tests:

In [2]:
```
!python test_knn.py
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.007s

OK
```

Loading Moons, Circles, and Rocky_Ridge Datasets:

In [3]:
```
moons = np.loadtxt('moons.csv', delimiter=',', skiprows=1)
circles = np.loadtxt('circles.csv', delimiter=',', skiprows=1)
rocky_ridge = np.loadtxt('rocky_ridge.csv', delimiter=',', skiprows=1)
```

Splitting The Datasets into training and testing splits (with stratification):

In [4]:
```
moons_X_train, moons_X_test, moons_labels_train, moons_labels_test = train_te
    moons[:, 1:], moons[:, 0], test_size=0.3, stratify=moons[:, 0])

circles_X_train, circles_X_test, circles_labels_train, circles_labels_test =
    circles[:, 1:], circles[:, 0], test_size=0.3, stratify=circles[:, 0])

rr_X_train, rr_X_test, rr_labels_train, rr_labels_test = train_test_split(
    rocky_ridge[:, 1:], rocky_ridge[:, 0], test_size=0.3, stratify=rocky_ridg
```

Standardizing the features of the datasets:

In [5]:
```
scaler = StandardScaler()

moons_X_train = scaler.fit_transform(moons_X_train)
moons_X_test = scaler.transform(moons_X_test)

circles_X_train = scaler.fit_transform(circles_X_train)
circles_X_test = scaler.transform(circles_X_test)

rr_X_train = scaler.fit_transform(rr_X_train)
rr_X_test = scaler.transform(rr_X_test)
```

Fitting KNN models to the datasets:

In [6]:
```
moons_knn = KNN(3, 'mode')
circles_knn = KNN(3, 'mode')
rocky_ridge_knn = KNN(3, 'mode')
```

In [7]:
```
moons_knn.fit(moons_X_train, moons_labels_train)
circles_knn.fit(circles_X_train, circles_labels_train)
rocky_ridge_knn.fit(rr_X_train, rr_labels_train)
```

Evaluating KNN models on a grid of points:

In [8]:
```
points = np.linspace(-3, 3, 300)
x, y = np.meshgrid(points, points)

test_points = np.array([x.flatten(),y.flatten()]).T

moons_values = moons_knn.predict(test_points)
circles_values = circles_knn.predict(test_points)
rocky_ridge_values = rocky_ridge_knn.predict(test_points)
```

Plotting the Moon Dataset's KNN Predictions:

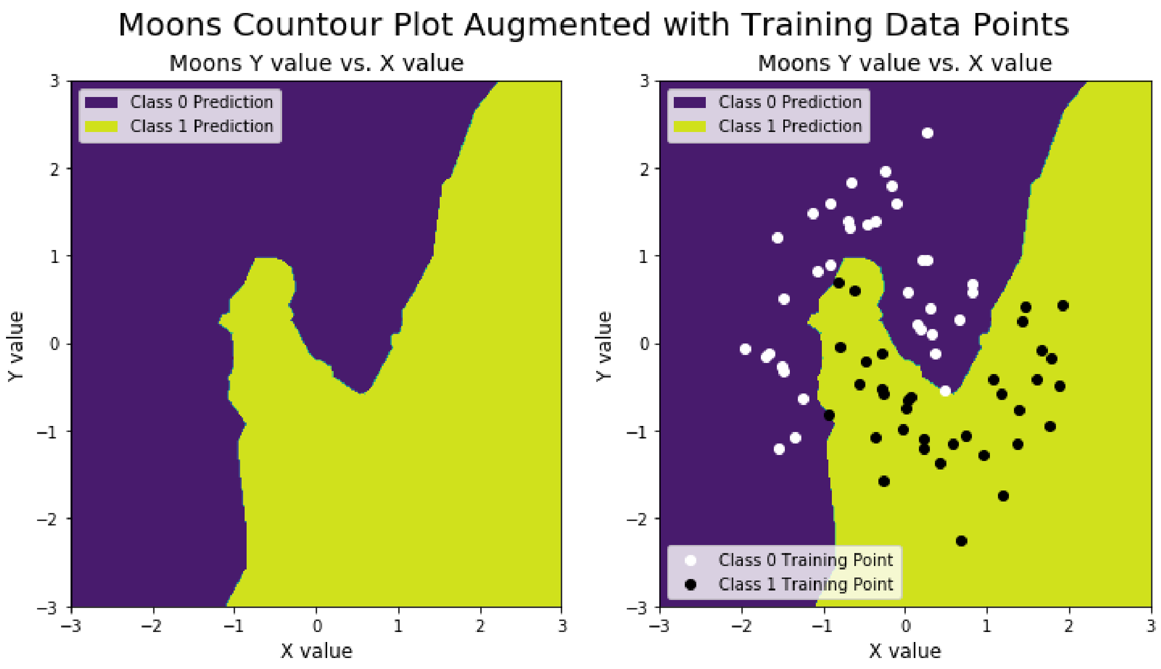
In [9]:
```
moons_class1_mask = moons_labels_train == 1
moons_X_train_class1 = moons_X_train[moons_class1_mask]
moons_X_train_class0 = moons_X_train[moons_class1_mask == False]
```

In [10]:
```
fig, axes = plt.subplots(1, 2, figsize=(12,6))
fig.suptitle('Moons Countour Plot Augmented with Training Data Points', size=
cs = axes[0].contourf(x, y, moons_values.reshape(300, 300));
axes[0].set_title('Moons Y value vs. X value', fontsize=14)
axes[0].set_xlabel('X value', fontsize=12)
axes[0].set_ylabel('Y value', fontsize=12)
proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
    for pc in cs.collections]
axes[0].legend([proxy[0], proxy[-1]], ['Class 0 Prediction', 'Class 1 Predict

cs = axes[1].contourf(x, y, moons_values.reshape(300, 300));
axes[1].scatter(moons_X_train_class0[:, 0], moons_X_train_class0[:, 1],
                c='w', label='Class 0 Training Point');
axes[1].scatter(moons_X_train_class1[:, 0], moons_X_train_class1[:, 1],
                c='k', label='Class 1 Training Point');
axes[1].set_title('Moons Y value vs. X value', fontsize=14)
axes[1].set_xlabel('X value', fontsize=12)
axes[1].set_ylabel('Y value', fontsize=12);

proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
    for pc in cs.collections]
first_legend = axes[1].legend(loc = 3)
plt.gca().add_artist(first_legend)
axes[1].legend([proxy[0], proxy[-1]], ['Class 0 Prediction', 'Class 1 Predict
```

Out[10]:
```
<matplotlib.legend.Legend at 0x7fd4efbb7da0>
```



Plotting the Circles Dataset's KNN Predictions:

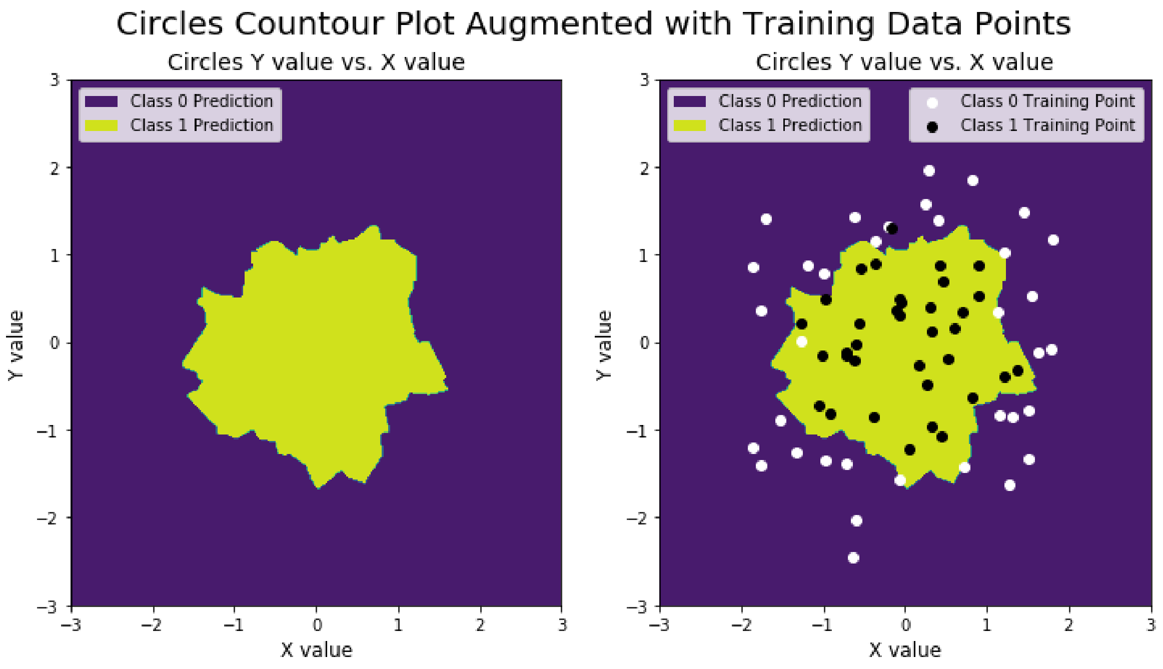
In [11]:
```
circles_class1_mask = circles_labels_train == 1
circles_X_train_class1 = circles_X_train[circles_class1_mask]
circles_X_train_class0 = circles_X_train[circles_class1_mask == False]
```

In [12]:
```
fig, axes = plt.subplots(1, 2, figsize=(12,6))
fig.suptitle('Circles Countour Plot Augmented with Training Data Points', siz
cs = axes[0].contourf(x, y, circles_values.reshape(300, 300));
axes[0].set_title('Circles Y value vs. X value', fontsize=14)
axes[0].set_xlabel('X value', fontsize=12)
axes[0].set_ylabel('Y value', fontsize=12)
proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
    for pc in cs.collections]
axes[0].legend([proxy[0], proxy[-1]], ['Class 0 Prediction', 'Class 1 Predict

cs = axes[1].contourf(x, y, circles_values.reshape(300, 300));
axes[1].scatter(circles_X_train_class0[:, 0], circles_X_train_class0[:, 1],
                c='w', label = 'Class 0 Training Point');
axes[1].scatter(circles_X_train_class1[:, 0], circles_X_train_class1[:, 1],
                c='k', label = 'Class 1 Training Point');
axes[1].set_title('Circles Y value vs. X value', fontsize=14)
axes[1].set_xlabel('X value', fontsize=12)
axes[1].set_ylabel('Y value', fontsize=12);

proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
    for pc in cs.collections]
first_legend = axes[1].legend(loc=0)
plt.gca().add_artist(first_legend)
axes[1].legend([proxy[0], proxy[-1]], ['Class 0 Prediction', 'Class 1 Predict
```

Out[12]:
```
<matplotlib.legend.Legend at 0x7fd4efa66be0>
```



Plotting the Rocky Ridge Dataset's KNN Predictions:

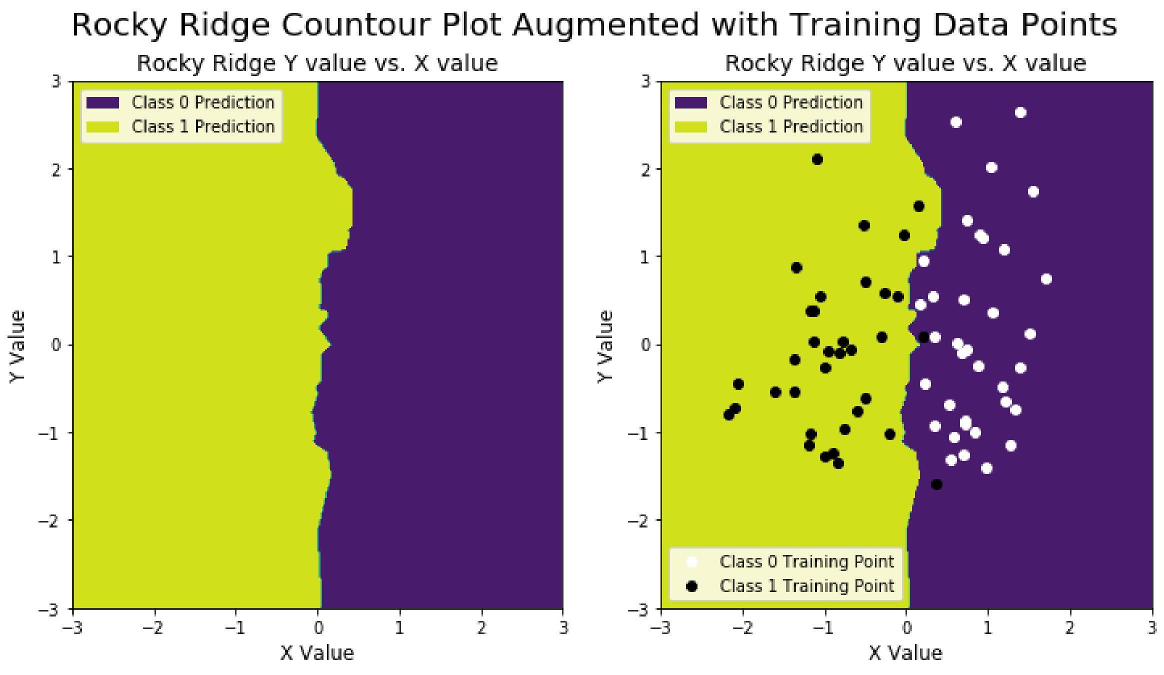
In [13]:
```
rr_class1_mask = rr_labels_train == 1
rr_X_train_class1 = rr_X_train[rr_class1_mask]
rr_X_train_class0 = rr_X_train[rr_class1_mask == False]
```

In [14]:
```
fig, axes = plt.subplots(1, 2, figsize=(12,6))
fig.suptitle('Rocky Ridge Countour Plot Augmented with Training Data Points',
cs = axes[0].contourf(x, y, rocky_ridge_values.reshape(300, 300));
axes[0].set_title('Rocky Ridge Y value vs. X value', fontsize=14)
axes[0].set_xlabel('X Value', fontsize=12)
axes[0].set_ylabel('Y Value', fontsize=12)
proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
    for pc in cs.collections]
axes[0].legend([proxy[0], proxy[-1]], ['Class 0 Prediction', 'Class 1 Predict

cs = axes[1].contourf(x, y, rocky_ridge_values.reshape(300, 300));
axes[1].scatter(rr_X_train_class0[:, 0], rr_X_train_class0[:, 1],
                c='w', label='Class 0 Training Point');
axes[1].scatter(rr_X_train_class1[:, 0], rr_X_train_class1[:, 1],
                c='k', label='Class 1 Training Point');
axes[1].set_title('Rocky Ridge Y value vs. X value', fontsize=14)
axes[1].set_xlabel('X Value', fontsize=12)
axes[1].set_ylabel('Y Value', fontsize=12);

proxy = [plt.Rectangle((0,0),1,1,fc = pc.get_facecolor()[0])
    for pc in cs.collections]
first_legend = axes[1].legend(loc = 3)
plt.gca().add_artist(first_legend)
axes[1].legend([proxy[0], proxy[-1]], ['Class 0 Prediction', 'Class 1 Predict
```

Out[14]:
```
<matplotlib.legend.Legend at 0x7fd4ef96d588>
```



Loading Sweep Dataset:

In [15]:
```
sweep_data = np.loadtxt('sweep.csv', delimiter=',', skiprows=1)
sweep_X = sweep_data[:, 1:]
sweep_labels = sweep_data[:, 0]
```

Evaluating effect of multiple k-values on Sweep KNN model:

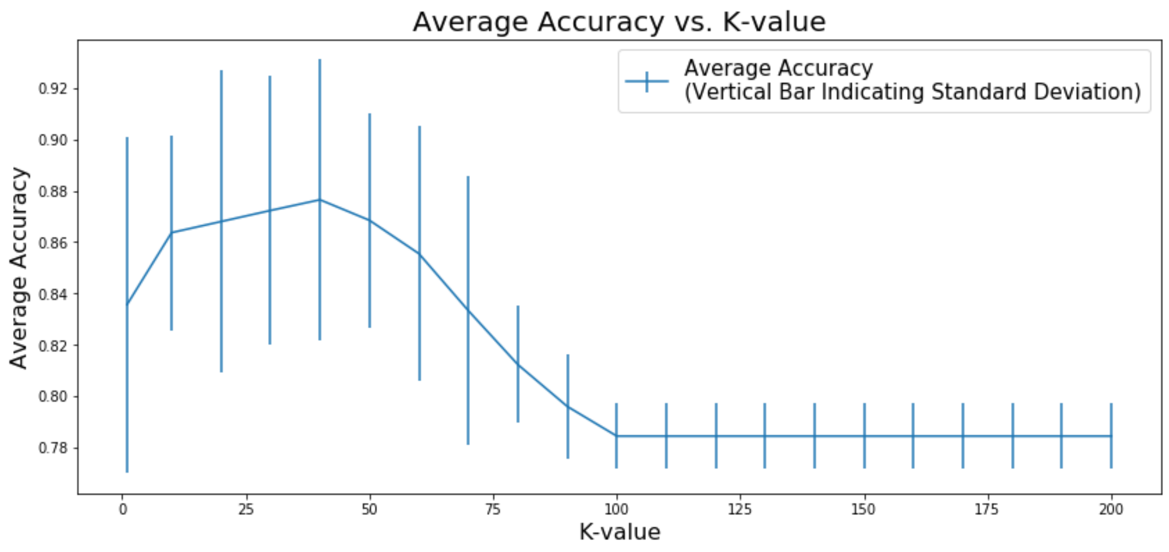
In [16]:
```
averages_accuracies = []
stds = []
k_values = list(range(10, 201, 10))
k_values.insert(0, 1)

for k in k_values:
    k = 1 if k is 0 else k
    sweep_knn = KNN(k, 'mode')
    accuracies = []
    skf = StratifiedKFold(n_splits=10, shuffle=True)
    for sweep_train_index, sweep_test_index in skf.split(sweep_X, sweep_labe
        sweep_knn.fit(sweep_X[sweep_train_index], sweep_labels[sweep_train_i
        predicted_labels = sweep_knn.predict(sweep_X[sweep_test_index])
        accuracies.append(accuracy_score(sweep_labels[sweep_test_index], pre
    averages_accuracies.append(np.mean(accuracies))
    stds.append(np.std(accuracies))
```

Plotting Average Accuracies vs. k-values:

In [17]:
```
fig, ax = plt.subplots(figsize=(14,6))
ax.errorbar(k_values, averages_accuracies, stds,
            label= 'Average Accuracy \n(Vertical Bar Indicating Standard Devi
ax.set_xlabel("K-value", fontsize = 16)
ax.set_ylabel("Average Accuracy", fontsize = 16)
ax.set_title("Average Accuracy vs. K-value", fontsize = 20)
ax.legend(prop={'size': 15})
```

Out[17]:
```
<matplotlib.legend.Legend at 0x7fd4ef88a048>
```


Average Accuracy vs. K-value
Average Accuracy
(Vertical Bar Indicating Standard Deviation)
Average Accuracy
K-value