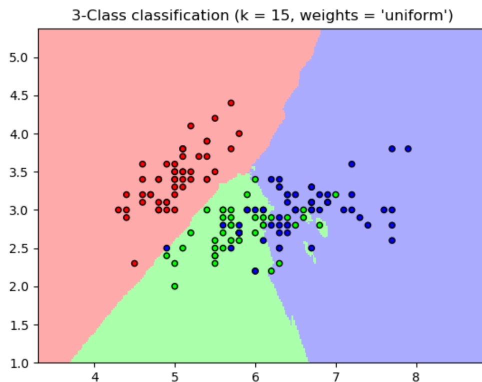
Machine Learning Activity
Nearest Neighbors in Scikit Learn

Content Learning Objectives

After completing this activity, students should be able to:

- Use python to change categorical data to numeric data
- Apply k Nearest Neighbors using the scikit-learn toolkit
- apply K-Fold cross validation
- tune the parameter k

Activity 1 : Nearest Neighbors – A Lazy Classifier



The k-nearest neighbors method (KNN) accepts data that consists of a set of features X (m by d matrix) and a class label C (a m by 1 matrix). Where each class label $c \in ClassLabels$, and ClassLabels is a finite discrete set. Recall that each row in the matrix is referred to as lower case x. KNN performs classification, that is, it takes a query point q and predicts the class label c for that q. When can envision this a $f(x) = q$, that is, there exist some function f that given the features x computes an estimate of q.

The figure above shows the application of KNN to a problems with 3 class labels (the size of the ClassLabels set is 3). When computing the label for a new point q (for which we do not know the class), KNN identifies the k closest points. Each of these points casts a vote based on its class. The class with the highest vote count is used to label q. To visualize the decision boundary, we can compute the label for a large set of q points (imagine constructing a grid of q points). The colors shown in the figure above represent how a point q would be classified and allow us to see the decision boundaries.

1. Compare the decision boundaries of KNN and decision trees. Be specific about what is similar and what is different.

2. What types of data (nominal, ordinal, interval, ratio) would be difficult to apply KNN? Why?

3. For each data type identified that would be “challenging”, how could you make KNN work?

Activity 2 : Using KNN from Scikit-Learn

Scikit learn provides some methods to take nominal and original data and transform them into numerical values.

```
from sklearn import preprocessing

# Assigning features and label variables
# First Feature
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny',
    'Sunny', 'Rainy','Sunny','Overcast','Overcast','Rainy']
# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild',
    'Mild','Mild','Hot','Mild']

# Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes',
    'Yes','No']
le = preprocessing.LabelEncoder()

weather_encoded = le.fit_transform(weather)
```

4. Write code to also transform the *temp* and *play* data.

5. What problems do you see with using the LabelEncoder? Print the *weather_encoded* variable?

Lets create a KNN classier to predict whether the person will go out to play (using the above data).

6. What are the features we will use? What is the class and class labels?

7. Python provides a function, zip, that accepts two lists and outputs a list of tuples. For example:

```
features = list(zip (myfeature1, myfeature2))
```

To build a KNN classifier in Scikit-learn, we will use the *KNeighborsClassifier* object (see `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`.

```
model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)
```

KFold Cross Validation

For evaluating the performance of a classifier, one approach is to separate the data you have into a *training* and *test* set. However, it can be difficult to decide what the ratio should be between the sets. Outliers and other factors could greatly influence the results of how the classifier is built and the reported accuracy.

Another approach is known as *k-fold* cross validation. In this approach, the data is divided up in to k equally sized pieces. One of these pieces is saved for test data and the remaining $k-1$ pieces are used for training. You then buid your model k times, rotating the piece you use for testing as shown in the Fig. You can judge the performance of the classifier on its average accuracy and standard dev.

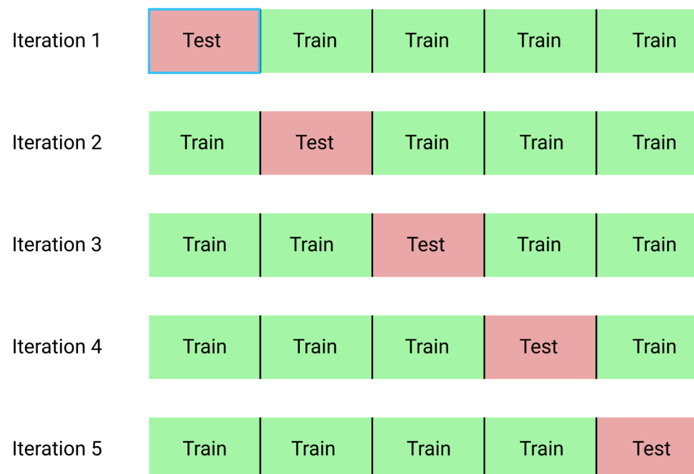


Figure 1: K-fold x validation

```
from sklearn import datasets
from sklearn import metrics
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import numpy as np

wine = datasets.load_wine()
k_fold = KFold(n_splits=3)

for k,(train,test) in enumerate(k_fold.split(wine.data,wine.target)):
    print(train.shape)
    print(test.shape)
```

Notice the different when you change the *KFold* call to:

```
k_fold = KFold(n_splits=3,shuffle=True)
```

SCIKIT-Learn has a method named *cross_val_score* that will run your classifier k times and gather the statistics for you. Here is an example call:

```
scores = cross_val_score(knn,X,y,cv=10,scoring='accuracy')
```

What about picking k?

8. Write python code that will evaluate 10 different values of k (1 - 10). To evaluate each method, use cross validation (with cv =10). Print and plot the accuracy of each K value. Here is some starter code.

```
k_range = range(1,11)
k_scores = []

for k in k_range:
   # build classifier here
   score = # validate classifier here using cross_val_score
    print ('k:',k,' mean:',scores.mean(), ' std:', scores.std())
    k_scores.append(scores.mean())

plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.plot()
```