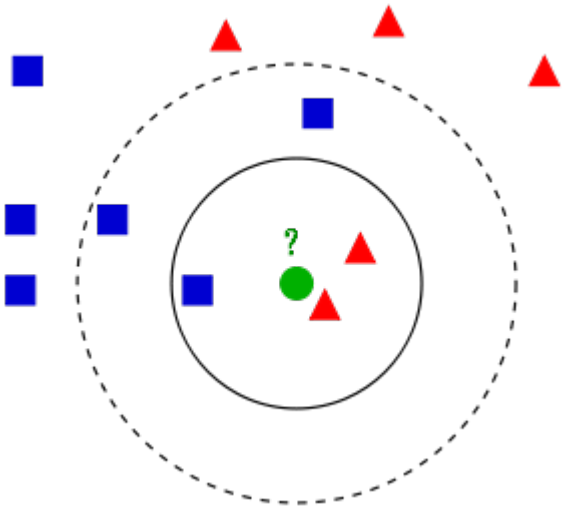
k-NN Classification

k-NN - k Nearest Neighbors

Neighbors-based classification is a type of *instance-based learning* or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data.

Classification is computed from a *simple majority vote* of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

k-NN Classification



The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.

If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

k-NN Classification

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is the Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the Hamming distance.

k-NN Classification

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

# get data
df = pd.read_csv("iris.csv")
X  = df.drop(['id','Species'],axis=1)
y = df['Species']

# set up the model
model = KNeighborsClassifier(n_neighbors=3)

# do train-test
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8)
model.fit(train_X, train_y)
predict_y = model.predict(test_X)
print("Train-Test Accuracy: {}".format(accuracy_score(test_y, predict_y)))

# do the 5-fold cross validation
scores = cross_val_score(model, X, y, cv=5)
print("Fold Accuracies: {}".format(scores))
print("XV Accuracy: {}".format(scores.mean()))
```

```
Train-Test Accuracy: 1.0
Fold Accuracies: [ 0.97  0.97  0.93  0.97  1.0  ]
XV Accuracy: 0.97
```

Model Comparison

We now have two different models we can use to do classification

Let’s work our way through an example using the dataset ‘wdbc’

Wdbc Exercise

Build optimal tree and KNN models using grid search

Compute the accuracy for the classifier

Print out the confusion matrix for each classifier

Print out the confidence interval for each classifier

Decide if the difference between classifiers is statistically significant or not.

Set Up

```
# basic data routines
import pandas as pd

# models
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

# model evaluation routines
from bootstrap import bootstrap
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

# get data
df = pd.read_csv(”wdbc.csv")
df = df.drop([’ID'],axis=1)
X  = df.drop([’Diagnosis'],axis=1)
actual_y = df[‘Diagnosis']
```

Trees

```
# decision trees
model = DecisionTreeClassifier()

# grid search
param_grid = {'max_depth': list(range(1,31)), 'criterion': ['entropy','gini'] }
grid = GridSearchCV(model, param_grid, cv=5)
grid.fit(X, actual_y)
print("Grid Search: best parameters: {}".format(grid.best_params_))

# evaluate the best model
best_model = grid.best_estimator_
predict_y = best_model.predict(X)
print("Accuracy: {}".format(accuracy_score(actual_y, predict_y)))

# build the confusion matrix
labels = [‘B', ’M']
cm = confusion_matrix(actual_y, predict_y, labels=labels)
cm_df = pd.DataFrame(cm, index=labels, columns=labels)
print("Confusion Matrix:\n{}".format(cm_df))

# boostrapped confidence interval
print("Confidence interval best decision tree: {}".format(bootstrap(best_model,df,’Diagnosis')))
```

Trees

```
Grid Search: best parameters: {'criterion': 'gini', 'max_depth': 5}

Accuracy: 0.9824304538799414

Confusion Matrix:
          B          M
B       436          8
M         4        235

Confidence interval best decision tree: (0.93430656934306566, 0.99270072992700731)
```

KNN

```
# KNN
model = KNeighborsClassifier()

# grid search
param_grid = {'n_neighbors': list(range(1,31))}
grid = GridSearchCV(model, param_grid, cv=5)
grid.fit(X, actual_y)
print("Grid Search: best parameters: {}".format(grid.best_params_))

# evaluate the best model
best_model = grid.best_estimator_

predict_y = best_model.predict(X)
print("Accuracy: {}".format(accuracy_score(actual_y, predict_y)))

# build the confusion matrix
labels = ['B', 'M']
cm = confusion_matrix(actual_y, predict_y, labels=labels)
cm_df = pd.DataFrame(cm, index=labels, columns=labels)
print("Confusion Matrix:\n{}".format(cm_df))

# boostrapped confidence interval
print("Confidence interval best KNN: {}".format(bootstrap(best_model,df,'Diagnosis')))
```

KNN

```
Grid Search: best parameters: {'n_neighbors': 5}

Accuracy: 0.9795021961932651

Confusion Matrix:
        B          M
B     434         10
M       4        235

Confidence interval best KNN: (0.94160583941605835, 1.0)
```