CSE 4950/6950
KNN

Instructor: AKM Kamrul Islam
aislam5@cs.gsu.edu
Dept. of Computer Science
Georgia State University

(Materials are highly adapted from different online sources)

Classification

- Assume we want to teach a computer to distinguish between cats and dogs …




Several steps:

1. feature transformation
2. Model / classifier specification
3. Model / classifier estimation (with regularization)
4. feature selection

Classification

- Assume we want to teach a computer to distinguish between cats and dogs …




Several steps:

1. feature transformation
2. Model / classifier specification
3. Model / classifier estimation (with regularization)
4. feature selection

How do we encode the picture? A collection of pixels? Do we use the entire image or a subset? …

Classification

- Assume we want to teach a computer to distinguish between cats and dogs …




Several steps:

1. feature transformation
2. Model / classifier specification
3. Model / classifier estimation (with regularization)
4. feature selection

What type of classifier should we use?

Classification

- Assume we want to teach a computer to distinguish between cats and dogs …

Several steps:

1. feature transformation
2. Model / classifier specification
3. Model / classifier estimation (with regularization)
4. feature selection

How do we learn the parameters of our classifier? Do we have enough examples to learn a good model?

Classification

- Assume we want to teach a computer to distinguish between cats and dogs …




Several steps:

1. feature transformation
2. Model / classifier specification
3. Model / classifier estimation (with regularization)
4. feature selection

Do we really need all the features? Can we use a smaller number and still achieve the same (or better) results?

Types of Classifier

- We can divide the large variety of classification approaches into roughly two main types

1. Instance based classifiers
   - Use observation directly (no models)
   - e.g. K nearest neighbors

2. Generative:
   - build a generative statistical model
   - e.g., Bayesian networks

3. Discriminative
   - directly estimate a decision rule/boundary
   - e.g., decision tree

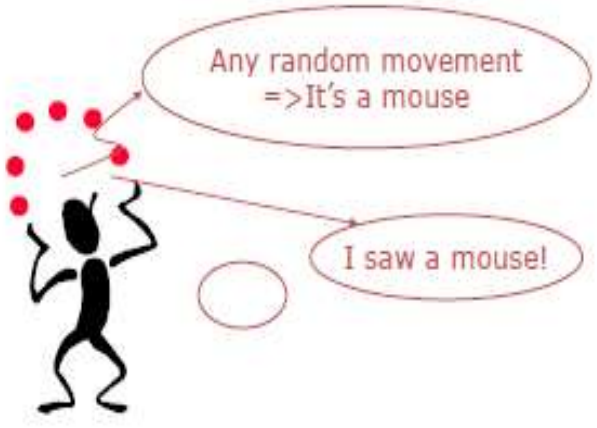
A classification of learning algorithms

- **Eager Learning**

– Learning = acquiring an explicit structure of a classifier on the whole training set;
– Classification = an instance gets a classification using the explicit structure of the classifier.

- **Instance-Based Learning (Lazy Learning)**

– Learning = storing all training instances
– Classification = an instance gets a classification equal to the classification of the nearest instances to the instance.



KNN

- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

  "Birds of a feather flock together."

- stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).
- KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

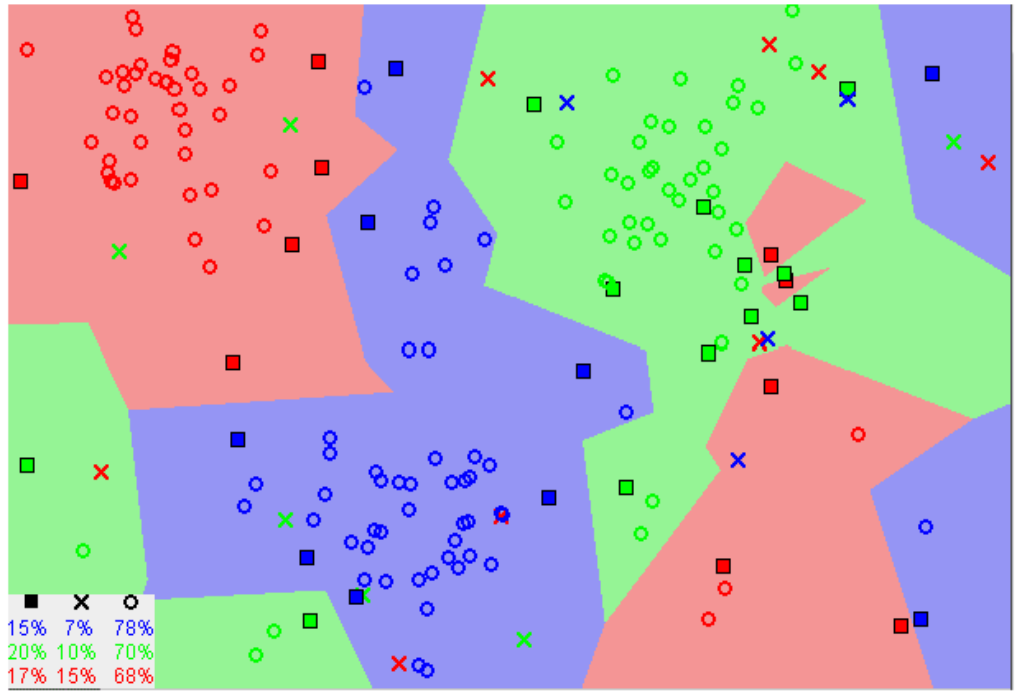
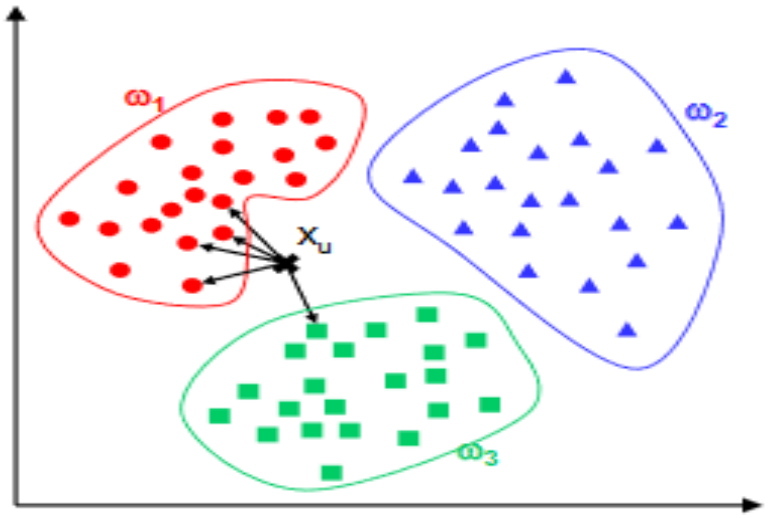


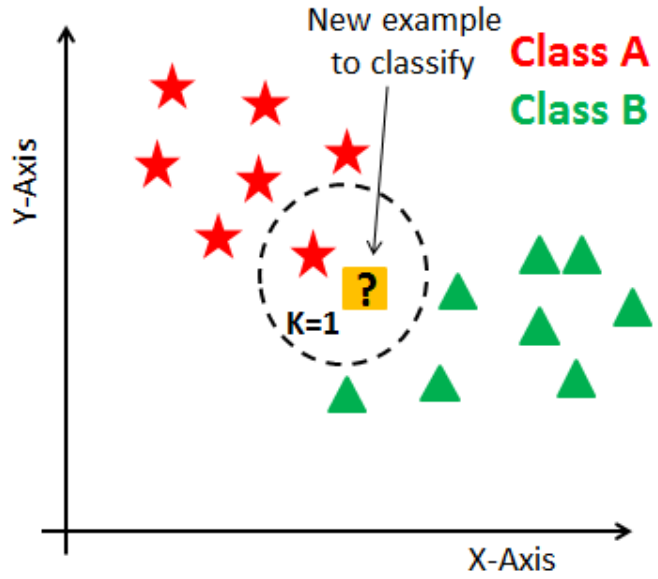
Image showing how similar data points typically exist close to each other

KNN

- The kNN rule is a very intuitive method that classifies unlabeled examples based on their similarity to examples in the training set
- For a given unlabeled example $x_u \in \Re^D$, find the k "closest" labeled examples in the training data set and assign x_u to the class that appears most frequently within the k-subset
- The kNN only requires
  - An integer k
  - A set of labeled examples (training data)
  - A metric to measure "closeness"
- Example
  - In the example here we have three classes and the goal is to find a class label for the unknown example x_u
  - In this case we use the Euclidean distance and a value of $k = 5$ neighbors
  - Of the 5 closest neighbors, 4 belong to ω_1 and 1 belongs to ω_3, so x_u is assigned to ω_1, the predominant class



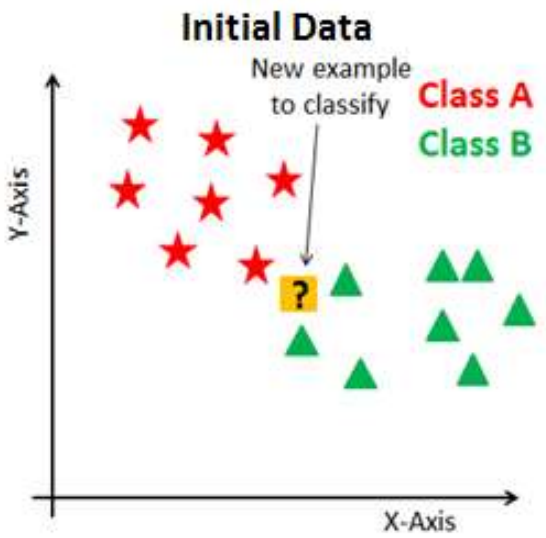
- In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2.
- When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case.
- Suppose P1 is the point, for which label needs to predict. First, you find the one closest point to P1 and then the label of the nearest point assigned to P1.

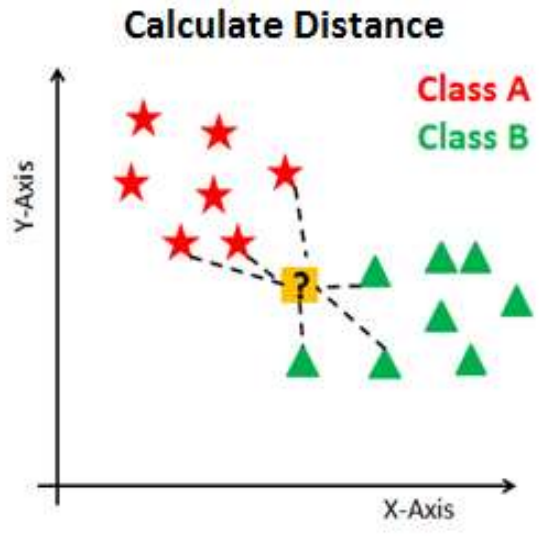


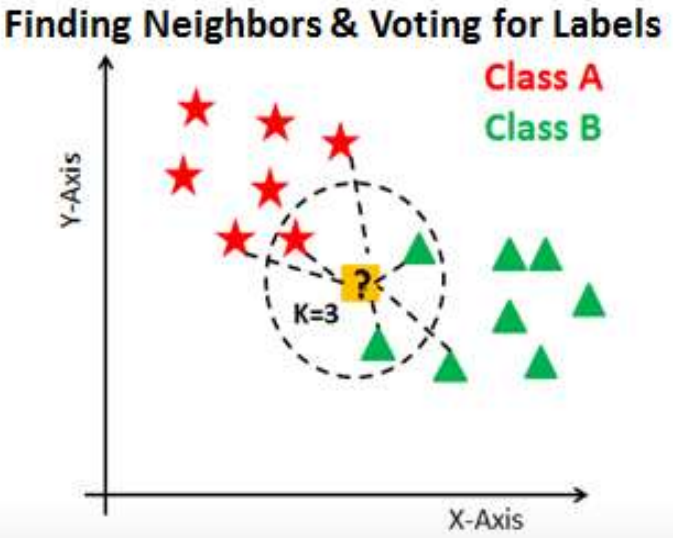
- Suppose P1 is the point, for which label needs to predict.
- First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors.
- Each object votes for their class and the class with the most votes is taken as the prediction.
- For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance.

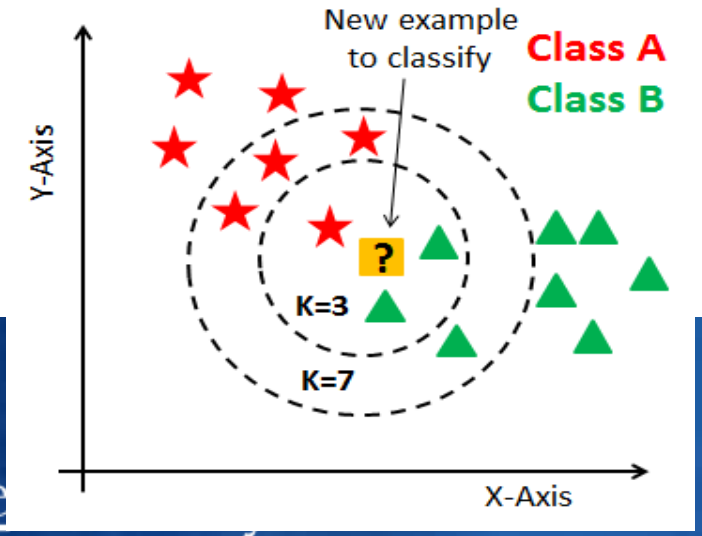
KNN has the following basic steps:
1.Calculate distance
2.Find closest neighbors
3.Vote for labels

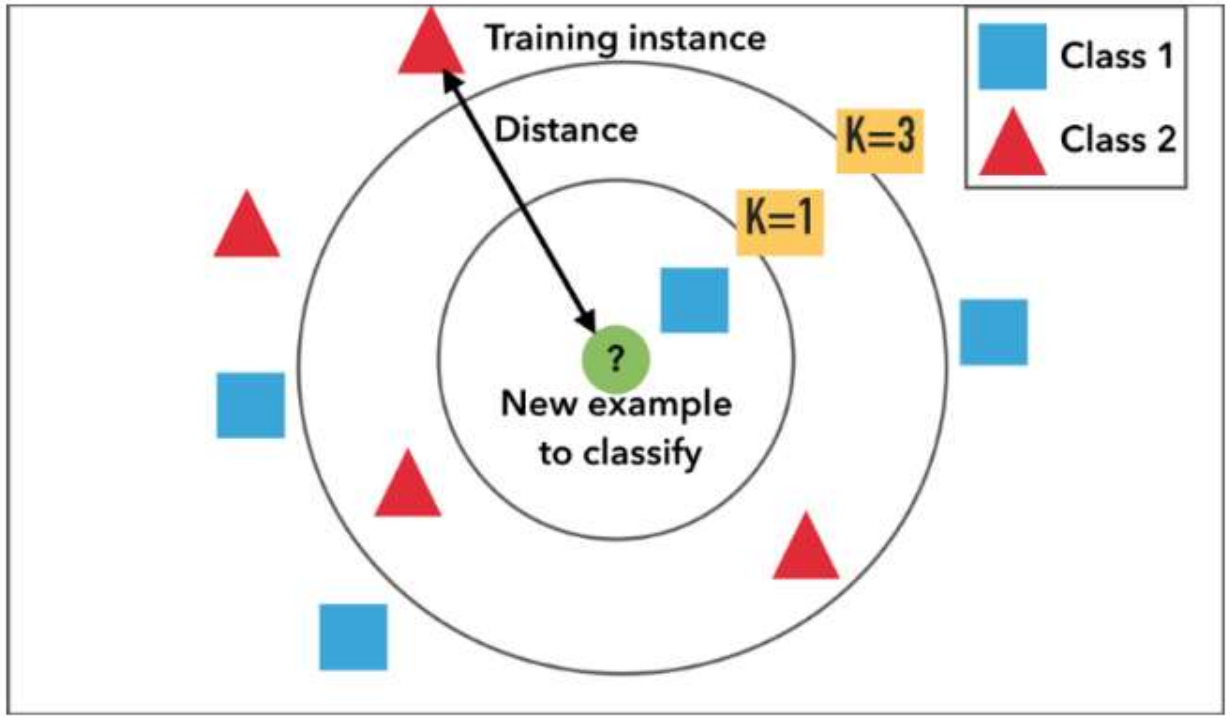
- Lazy Learning means there is no need for learning or training of the model and all of the data points used at the time of prediction.
- Lazy learners wait until the last minute before classifying any data point. Lazy learner stores merely the training dataset and waits until classification needs to perform.











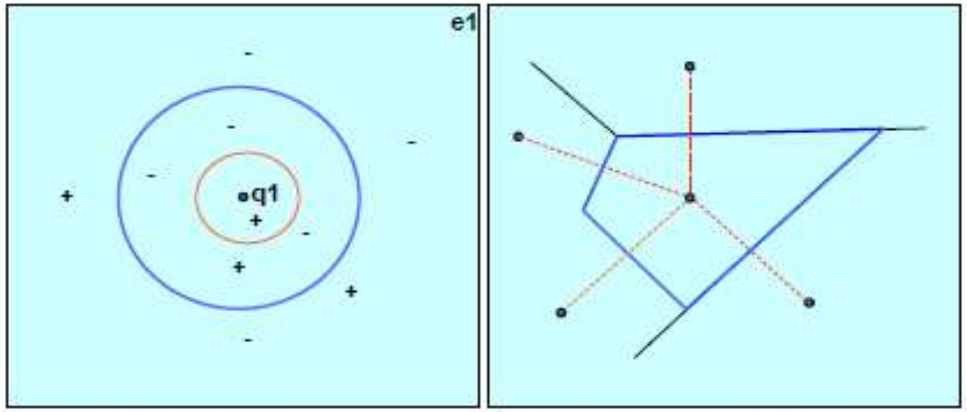
Example of *k*-NN classification.

- The test sample (inside circle) should be classified either to the first class of blue squares or to the second class of red triangles.
- If *k = 3* (outside circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If, for example *k = 5* it is assigned to the first class (3 squares vs. 2 triangles outside the outer circle).

Nearest-Neighbor Algorithm (NN)

- The Features of the Task of the NN Algorithm:
- the instance language *I* is a conjunctive language with a set *A* with *n* attributes *a1*, *a2*, … *an*. The domain of each attribute *ai*, can be discrete or continuous.
- an instance *x* is represented as < *a1*(*x*), *a2*(*x*), … *an*(*x*) >, where *ai*(*x*) is the value of the attribute *ai* for the instance *x*;
- the classes to be learned can be:
  - *discrete*. In this case we learn *discrete* function *f*(*x*) and the co-domain *C* of the function consists of the classes *c* to be learned.
  - *continuous*. In this case we learn *continuous* function *f*(*x*) and the co-domain *C* of the function consists of the classes *c* to be learned.

Classification & Decision Boundaries



Example

Training dataset

Customer ID	Debt	Income	Marital Status	Risk
Abel	High	High	Married	Good
Ben	Low	High	Married	Doubtful
Candy	Medium	Very low	Unmarried	Poor
Dale	Very high	Low	Married	Poor
Ellen	High	Low	Married	Poor
Fred	High	Very low	Married	Poor
George	Low	High	Unmarried	Doubtful
Harry	Low	Medium	Married	Doubtful
Igor	Very Low	Very High	Married	Good
Jack	Very High	Medium	Married	Poor

Test Set

Customer ID	Debt	Income	Marital Status	Risk
Zeb	High	Medium	Married	?
Yong	Low	High	Married	?
Xu	Very low	Very low	Unmarried	?
Vasco	High	Low	Married	?
Unace	High	Low	Divorced	?
Trey	Very low	Very low	Married	?
Steve	Low	High	Unmarried	?

K = 3

- Distance
  - Score for an attribute is 0 for a match and 1 otherwise
  - Distance is sum of scores for each attribute
- Voting scheme: proportionate voting in case of ties

Distance Weighted Nearest-Neighbor Algorithm

- Weight the contribution of each of the k neighbors according to their distance to the query point $\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$

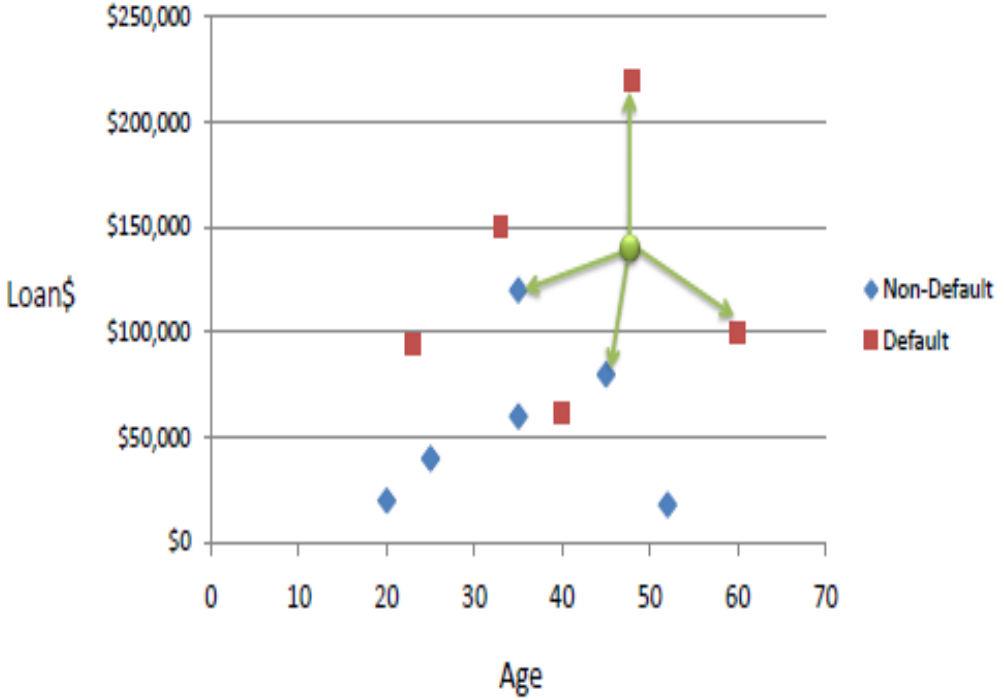
$$w_i = \frac{1}{d(x_q, x_i)^2}$$

Customer ID	Debt	Income	Marital Status	Risk
Abel	High	High	Married	Good
Ben	Low	High	Married	Doubtful
Candy	Medium	Very low	Unmarried	Poor
Dale	Very high	Low	Married	Poor
Ellen	High	Low	Married	Poor
Fred	High	Very low	Married	Poor
George	Low	High	Unmarried	Doubtful
Harry	Low	Medium	Married	Doubtful
Igor	Very Low	Very High	Married	Good
Jack	Very High	Medium	Married	Poor

Customer ID	Debt	Income	Marital Status	Risk
Zeb	High	Medium	Married	?
Yong	Low	High	Married	?
Xu	Very low	Very low	Unmarried	?
Vasco	High	Low	Married	?
Unace	High	Low	Divorced	?
Trey	Very low	Very low	Married	?
Steve	Low	High	Unmarried	?

Explanations KNN

- Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target
- We can now use the training set to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance. If K=1 then the nearest neighbor is the last case in the training set with Default=Y.
- With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y



D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01 >> Default=Y

Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	**$142,000**	**?**		

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Distance Functions

- A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.
- It should also be noted that all three distance measures are only valid for continuous variables.
- In the instance of categorical variables the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Hamming Distance

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

Distance functions

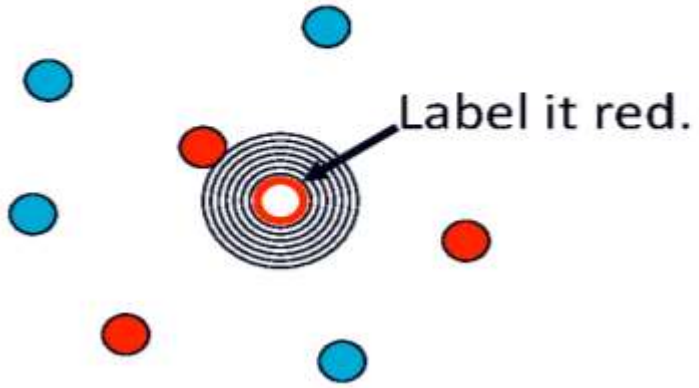
Euclidean $\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k} |x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k} (|x_i - y_i|)^q \right)^{1/q}$

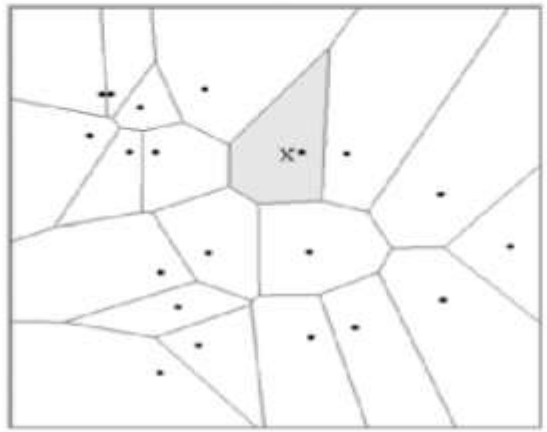
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point



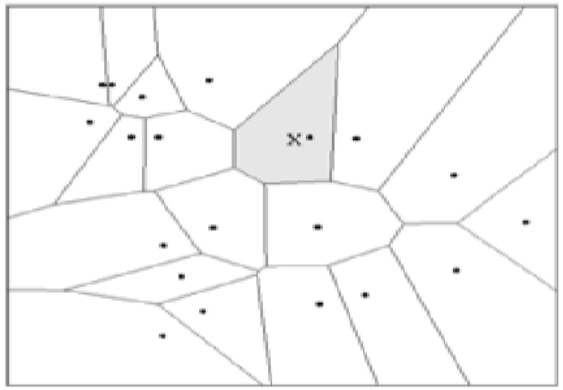
1-Nearest Neighbor

- A type of instance-based learning
  - Also known as "memory-based" learning
- Forms a Voronoi tessellation of the instance space

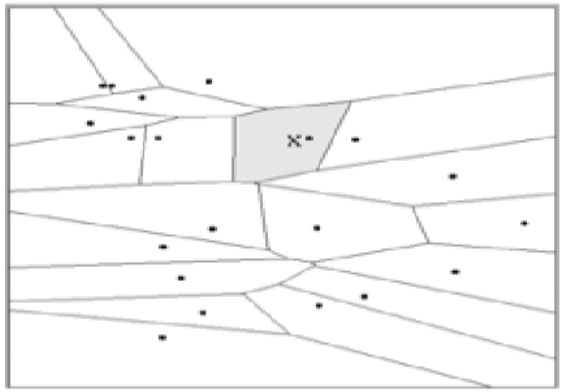


Distance Metrics

- Different metrics can change the decision surface



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (a_2 - b_2)^2$



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

- Standard Euclidean distance metric:
  - Two-dimensional: $\text{Dist}(a,b) = \text{sqrt}((a_1 - b_1)^2 + (a_2 - b_2)^2)$
  - Multivariate: $\text{Dist}(a,b) = \text{sqrt}(\sum (a_i - b_i)^2)$

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

Four Aspects of an Instance-Based Learner:

1. A distance metric
2. How many nearby neighbors to look at?
3. A weighting function (optional)
4. How to fit with the local points?

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

1-NN's Four Aspects as an Instance-Based Learner:

1. A distance metric
   - *Euclidian*
2. How many nearby neighbors to look at?
   - *One*
3. A weighting function (optional)
   - *Unused*
4. How to fit with the local points?
   - *Just predict the same output as the nearest neighbor.*

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

Zen Gardens

Mystery of renowned zen garden revealed [CNN Article]
Thursday, September 26, 2002 Posted: 10:11 AM EDT (1411 GMT)

LONDON (Reuters) -- For centuries visitors to the renowned Ryoanji Temple garden in Kyoto, Japan have been entranced and mystified by the simple arrangement of rocks.

The five sparse clusters on a rectangle of raked gravel are said to be pleasing to the eyes of the hundreds of thousands of tourists who visit the garden each year.

Scientists in Japan said on Wednesday they now believe they have discovered its mysterious appeal.

"We have uncovered the implicit structure of the Ryoanji garden's visual ground and have shown that it includes an abstract, minimalist depiction of natural scenery," said Gert Van Tonder of Kyoto University.

The researchers discovered that the empty space of the garden evokes a hidden image of a branching tree that is sensed by the unconscious mind.

"We believe that the unconscious perception of this pattern contributes to the enigmatic appeal of the garden," Van Tonder added.

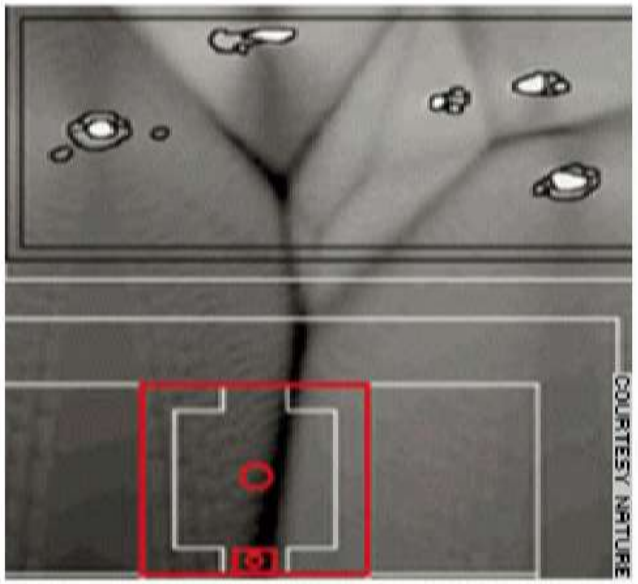
He and his colleagues believe that whoever created the garden during the Muromachi era between 1333-1573 knew exactly what they were doing and placed the rocks around the tree image.

By using a concept called medial-axis transformation, the scientists showed that the hidden branched tree converges on the main area from which the garden is viewed.

The trunk leads to the prime viewing site in the ancient temple that once overlooked the garden. It is thought that abstract art may have a similar impact.

"There is a growing realisation that scientific analysis can reveal unexpected structural features hidden in controversial abstract paintings," Van Tonder said

Ryoanji Temple garden in Kyoto

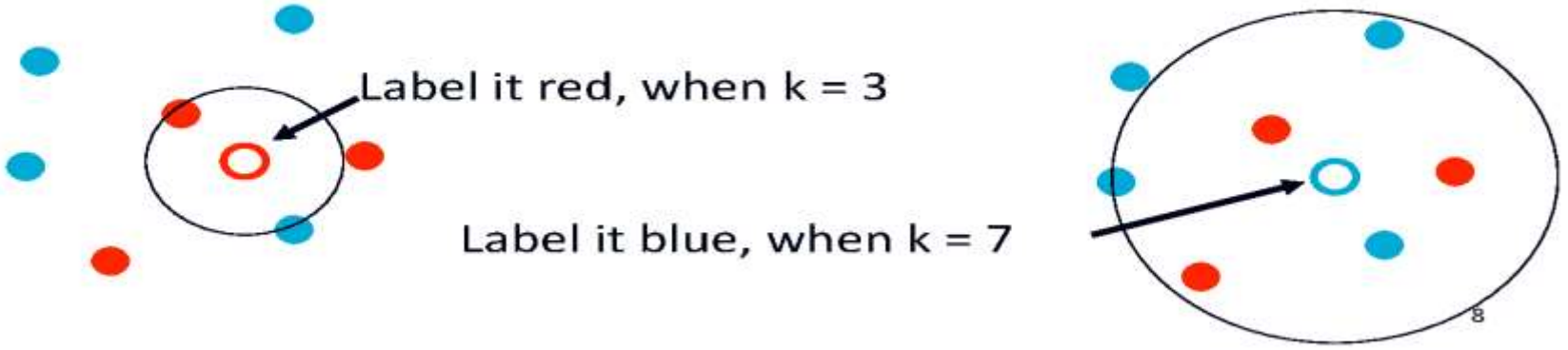


Layout shows the rock clusters (top) and the preferred viewing spot of the garden from the main hall (the circle in the middle of the square).

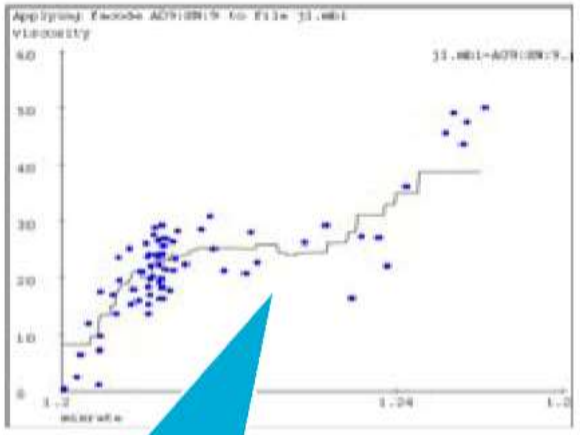
Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

k – Nearest Neighbor

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its *k* nearest neighbors

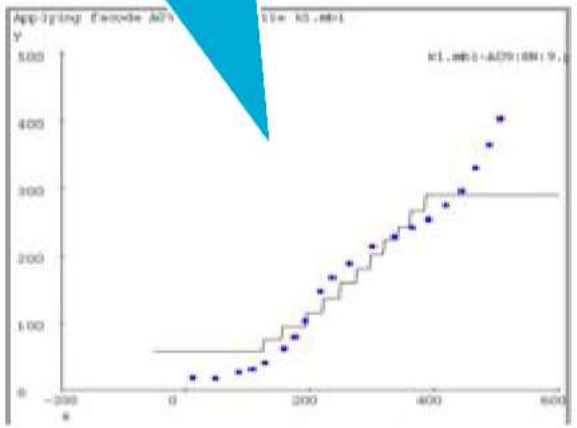


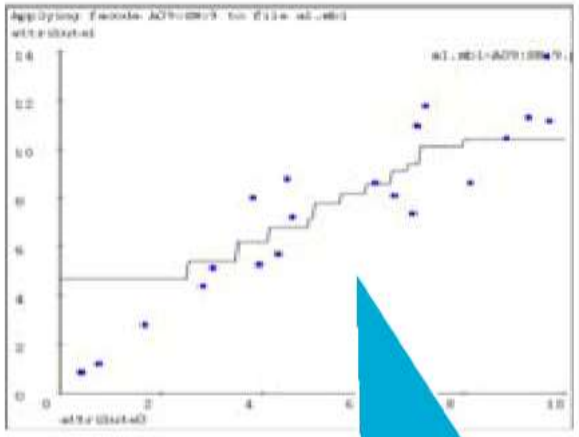
k-Nearest Neighbor (k = 9)



A magnificent job of noise smoothing. Three cheers for 9-nearest-neighbor.
...But the lack of gradients and the jerkiness isn't good.

Appalling behavior! Loses all the detail that 1-nearest neighbor would give. The tails are horrible!





Fits much less of the noise, captures trends. But still, frankly, pathetic compared with linear regression.

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

Choosing the right value for K

To select the K that's right for the data, run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors.

Here are some things to keep in mind:

1. As we decrease the value of K to 1, our predictions become less stable.
2. Inversely, increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point).
3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Advantages

1. The algorithm is simple and easy to implement.
2. There's no need to build a model, tune several parameters, or make additional assumptions.
3. The algorithm is versatile. It can be used for classification, regression, and search

Disadvantages

1. The algorithm gets significantly slower as the number of predictors/independent variables increase.
2. Computationally expensive — because the algorithm stores all of the training data
3. High memory requirement
4. Stores all (or almost all) of the training data
5. Prediction stage might be slow (with big N)

Applications

1. Recommending products on Amazon, articles on Medium, movies on Netflix, or videos on YouTube.
2. In political science — classing a potential voter to a "will vote" or "will not vote", or to "vote Democrat" or "vote Republican".
3. Handwriting detection (like OCR), image recognition and even video recognition

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbours
3. For each example in the data
    - Calculate the distance between the query example and the current example from the data.
    - Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
    - If regression, return the mean of the K labels
    - If classification, return the mode of the K labels

KNN Similarity based learning

- Needed: A feature space representation of the instance in the dataset and a measure of similarity between instances.
- Each instance in the training set is stored in a memory.
- Initial storing is standard however once all training examples are stored a second run needs to determine the distance between the different instances.
- The prediction is based on finding out what class the nearest instance belongs to.

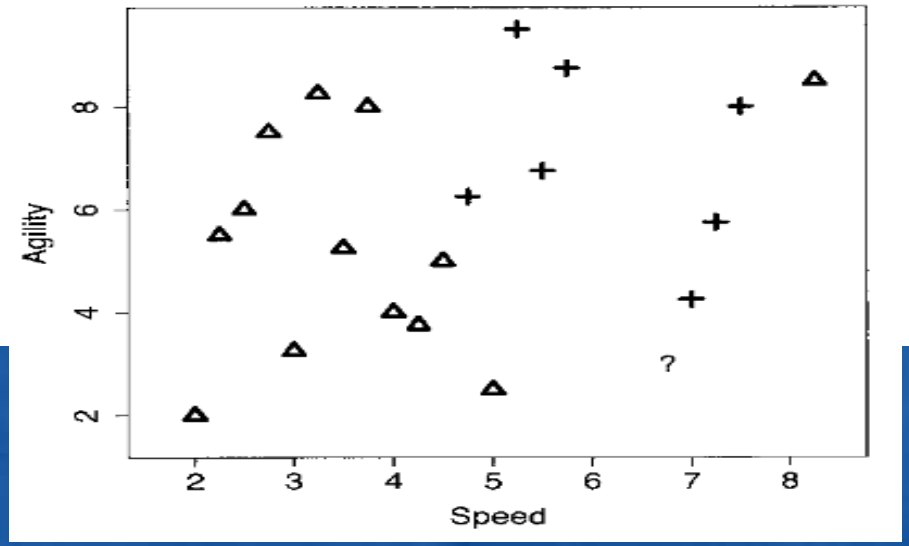
Require: a set of training instances
Require: a query instance

1. Iterate across the instances in memory to find the nearest neighbour –this is the instance with the shortest distance across the feature space to the query instance.
2. Make a prediction for the query instance that is equal to the value of the target feature of the nearest neighbour.

The SPEED and AGILITY ratings for 20 college athletes and whether they were drafted by a professional team.

ID	SPEED	AGILITY	DRAFT	ID	SPEED	AGILITY	DRAFT
1	2.50	6.00	no	11	2.00	2.00	no
2	3.75	8.00	no	12	5.00	2.50	no
3	2.25	5.50	no	13	8.25	8.50	no
4	3.25	8.25	no	14	5.75	8.75	yes
5	2.75	7.50	no	15	4.75	6.25	yes
6	4.50	5.00	no	16	5.50	6.75	yes
7	3.50	5.25	no	17	5.25	9.50	yes
8	3.00	3.25	no	18	7.00	4.25	yes
9	4.00	4.00	no	19	7.50	8.00	yes
10	4.25	3.75	no	20	7.25	5.75	yes

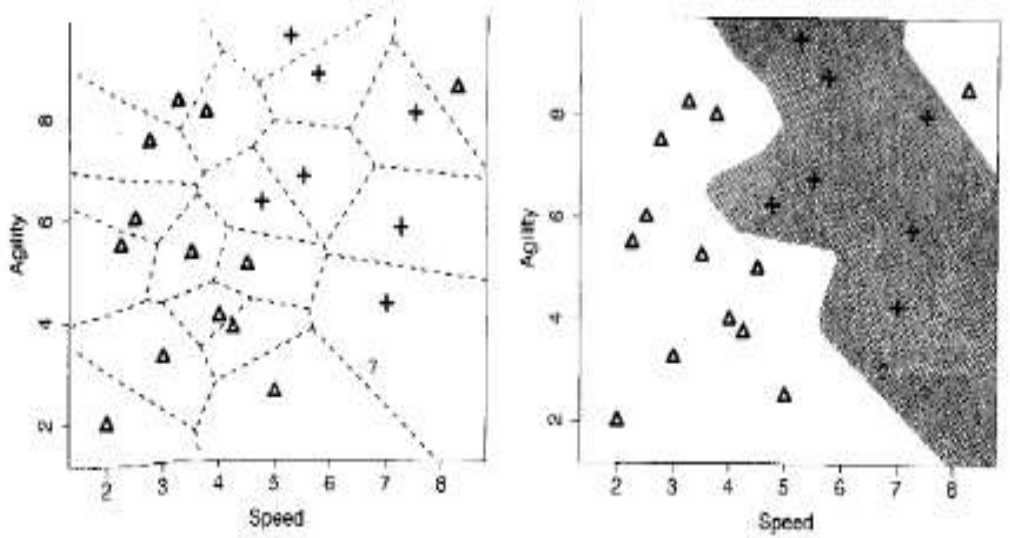
From: Fundamental of Machine Learning John Kelleher



- KNN creates local models (or neighbourhoods) across the feature space with each space defined by a subset of the training data.
- Implicitly a 'global' decision space is created with boundaries between the training data
- One advantage of KNN is that updating the decision space is easy.
- KNN is a nearest neighbour algorithm that creates an implicit global classification model by aggregating local models, or neighborhoods.

The distances (Dist.) between the query instance with SPEED = 6.75 and AGILITY = 3.00 and each instance in Table 5.2[182].

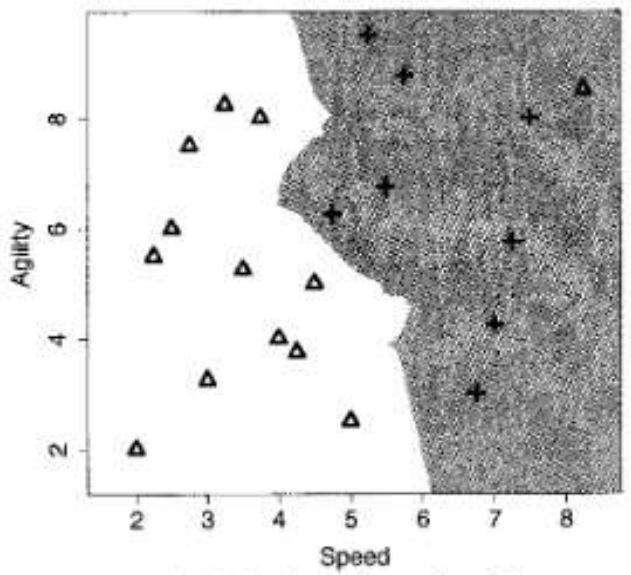
ID	SPEED	AGILITY	DRAFT	Dist.	ID	SPEED	AGILITY	DRAFT	Dist.
18	7.00	4.25	yes	1.27	11	2.00	2.00	no	4.85
12	5.00	2.50	no	1.82	19	7.50	8.00	yes	5.06
10	4.25	3.75	no	2.61	3	2.25	5.50	no	5.15
20	7.25	5.75	yes	2.80	1	2.50	6.00	no	5.20
9	4.00	4.00	no	2.93	13	8.25	8.50	no	5.70
6	4.50	5.00	no	3.01	2	3.75	8.00	no	5.83
8	3.00	3.25	no	3.76	14	5.75	8.75	yes	5.84
15	4.75	6.25	yes	3.82	5	2.75	7.50	no	6.02
7	3.50	5.25	no	3.95	4	3.25	8.25	no	6.31
16	5.50	6.75	yes	3.95	17	5.25	9.50	yes	6.67



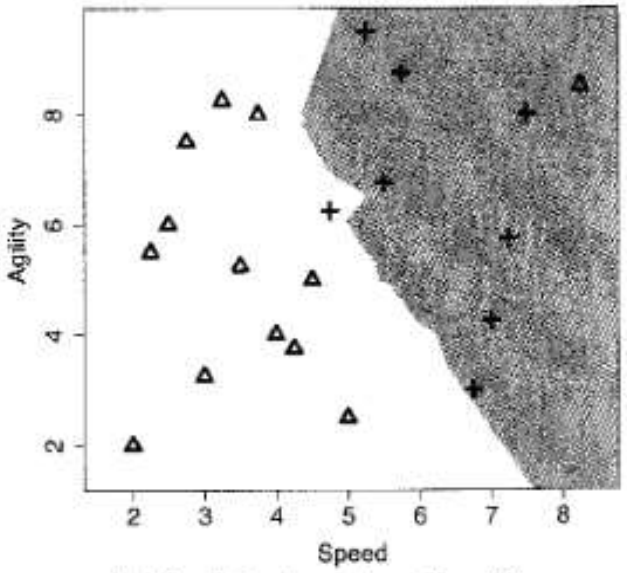
(a) Voronoi tessellation (b) Decision boundary ($k = 1$)

Handling noisy data

- Outliers can create individual spaces which belong to a class but are separated. This mostly relates to noise in the data
- The solution is dilution of dependency of the algorithm on individual (possibly noisy) instances.



(a) Decision boundary ($k = 3$)



(b) Decision boundary ($k = 5$)

Voting

- Once we have obtained the K-Nearest-Neighboursusing the distance function, it is time for the neighbors to vote in order to predict it's class class.
- Majority voting assumes that all votes are equal.
- For each class l ∈L we count the amount of k-neighboursthat have that class.
- We return the class with the most votes.

- Modification of the algorithm to return the majority vote within the set of k nearest neighbours to a query q.
- Mk(q) is the prediction of the model M for query q given the parameter of the model k.
- Levels(l) is the set of of levels (classes) in the domain of the target feature and l is an element of this set.
- i iterates over the distance d_i in increasing distance from the query q
- t_i is the value of the target feature for distance d_i
- Delta(t_i, l) is the KnoeckerDelta function which takes two parameters and returns 1 if they are equal or 0 if not.

$$\mathbb{M}_k(\mathbf{q}) = \arg\max_{l \in levels(t)} \sum_{i=1}^{k} \delta(t_i, l)$$

In [1]:

```
#Load the necessary python libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [2]:

```
#Load the dataset
df = pd.read_csv('../input/diabetes.csv')

#Print the first 5 rows of the dataframe.
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [3]:

```
#Let's observe the shape of the dataframe.
df.shape
```

Out[3]:

```
(768, 9)
```

- As observed above we have 768 rows and 9 columns. The first 8 columns represent the features and the last column represent the target/label

In [4]:

```
#Let's create numpy arrays for features and target
X = df.drop('Outcome',axis=1).values
y = df['Outcome'].values
```

- Let's split the data randomly into training and test set.
- We will fit/train a classifier on the training set and make predictions on the test set. Then we will compare the predictions with the known labels.
- Scikit-learn provides facility to split data into train and test set using train_test_split method

In [5]:
```
#importing train_test_split
from sklearn.model_selection import train_test_split
```

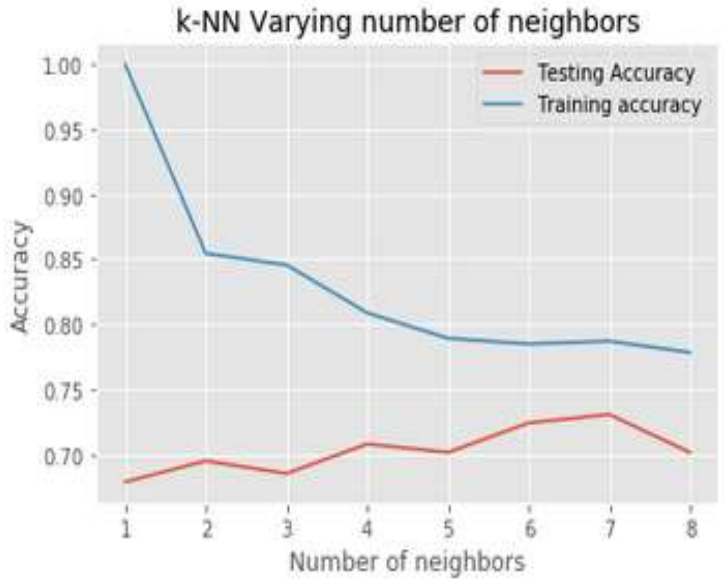
- It is a best practice to perform our split in such a way that out split reflects the labels in the data.
- In other words, we want labels to be split in train and test set as they are in the original dataset. So we use the stratify argument.
- Also we create a test set of size of about 40% of the dataset.

In [6]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_state=42, st
ratify=y)
```

- Let's create a classifier using k-Nearest Neighbors algorithm.
- First let us first observe the accuracies for different values of k.

In [7]:
```
#import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

#Setup arrays to store training and test accuracies
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

In [8]:
```
#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [9]:
```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=7)
```

In [10]:
```
#Fit the model
knn.fit(X_train,y_train)
```

Out[10]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=7, p=2,
           weights='uniform')
```

In [11]:
```
#Get accuracy. Note: In case of classification algorithms score method represents accuracy.
knn.score(X_test,y_test)
```

Out[11]:
```
0.7305194805194806
```

- We can observe above that we get maximum testing accuracy for k=7.
- So lets create a KNeighborsClassifier with number of neighbors as 7.

In [12]:
```
#import confusion_matrix
from sklearn.metrics import confusion_matrix
```

In [13]:
```
#let us get the predictions using the classifier we had fit above
y_pred = knn.predict(X_test)
```

In [14]:
```
confusion_matrix(y_test,y_pred)
```

Out[14]:
```
array([[165,  36],
       [ 47,  60]])
```

Considering confusion matrix above:

True negative = 165

False positive = 36

True postive = 60

Fasle negative = 47

In [15]:
```
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Out[15]:

Predicted	0	1	All
True			
0	165	36	201
1	47	60	107
All	212	96	308

In [16]:
```
#import classification_report
from sklearn.metrics import classification_report
```

In [17]:
```
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.78      0.82      0.80       201
          1       0.62      0.56      0.59       107

avg / total       0.73      0.73      0.73       308
```

Georgia State University

ROC (Receiver Operating Characteristic) curve

It is a plot of the true positive rate against the false positive rate for the different possible cut points of a diagnostic test.

An ROC curve demonstrates several things:

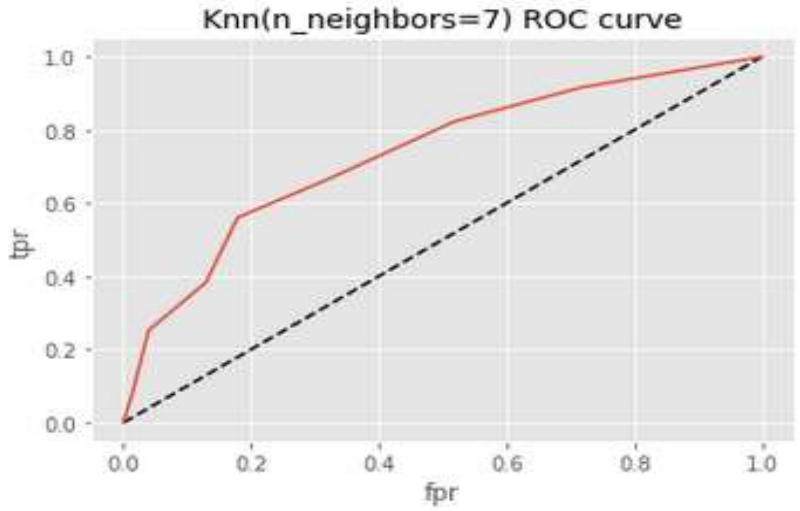
1. It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
2. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
3. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.
4. The area under the curve is a measure of test accuracy.

In [18]:
```
y_pred_proba = knn.predict_proba(X_test)[:,1]
```

In [19]:
```
from sklearn.metrics import roc_curve
```

In [20]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [21]:
```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=7) ROC curve')
plt.show()
```



In [22]:
```
#Area under ROC curve
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

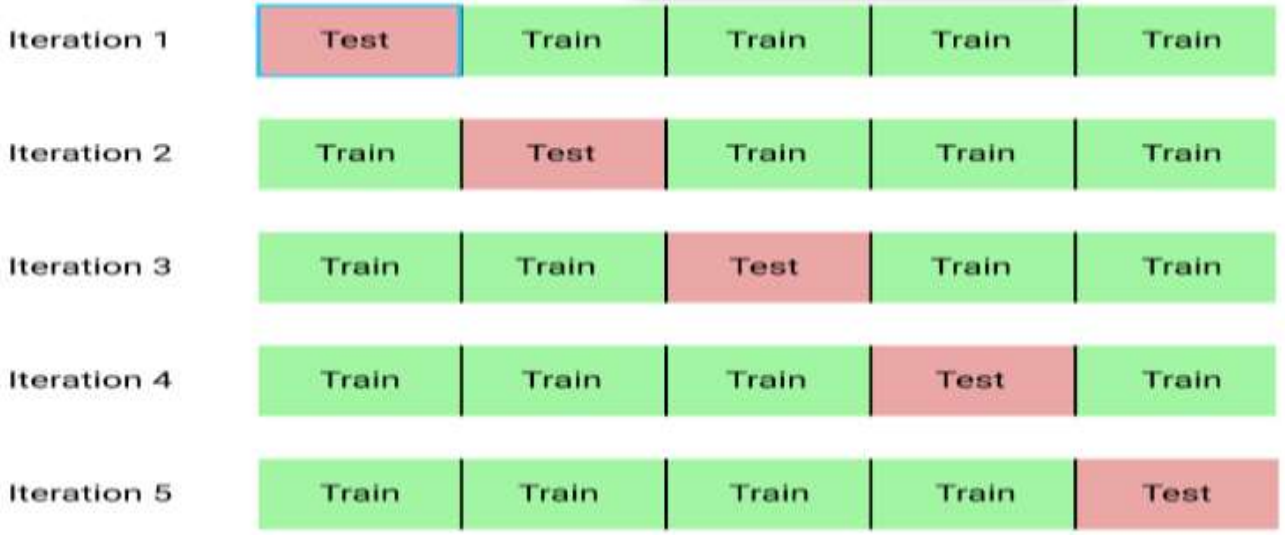
Out[22]:
```
0.7345050448691124
```

Hyperparameter tuning

- The value of k (i.e 7) we selected above was selected by observing the curve of accuracy vs number of neighbors.
- This is a primitive way of hyperparameter tuning.

There is a better way of doing it which involves:

1) Trying a bunch of different hyperparameter values
2) Fitting all of them separately
3) Checking how well each performs
4) Choosing the best performing one
5) Using cross-validation every time



In [23]:
```
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
```

In [24]:
```
#In case of classifier like knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':np.arange(1,50)}
```

In [25]:
```
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)
```

Out[25]:
```
GridSearchCV(cv=5, 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': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13,
14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [26]:
```
knn_cv.best_score_
```

Out[26]:
```
0.7578125
```

In [27]:
```
knn_cv.best_params_
```

Out[27]:
```
{'n_neighbors': 14}
```

- Thus a knn classifier with number of neighbors as 14 achieves the best score/accuracy of 0.7578 i.e about 76%

A probabilistic interpretation of KNN

The decision rule of KNN can be viewed using a probabilistic interpretation

- What KNN is trying to do is approximate the Bayes decision rule on a subset of the data
- To do that we need to compute certain properties including the conditional probability of the data given the class ($p(x|y)$), the prior probability of each class ($p(y)$) and the marginal probability of the data ($p(x)$)
- These properties would be computed for some small region around our sample and the size of that region will be *dependent on the distribution of the test samples*

Computing probabilities for KNN

- Let V be the volume of the m dimensional ball around z containing the k nearest neighbors for z (where m is the number of features).
- Then we can write

$$p(x)V = P = \frac{K}{N} \qquad p(x) = \frac{K}{NV} \qquad p(x \mid y=1) = \frac{K_1}{N_1 V} \qquad p(y=1) = \frac{N_1}{N}$$

- Using Bayes rule we get:

$$p(y=1 \mid z) = \frac{p(z \mid y=1)p(y=1)}{p(z)} = \frac{K_1}{K}$$

z – new data point to classify
V - selected ball
P – probability that a random point is in V
N - total number of samples
K - number of nearest neighbors
N_1 - total number of samples from class 1
K_1 - number of samples from class 1 in K

Computing probabilities for KNN

N - total number of samples
V - Volume of selected ball
K - number of nearest neighbors
N_1 - total number of samples from class 1
K_1 - number of samples from class 1 in K

- Using Bayes rule we get:

$$p(y=1 \mid z) = \frac{p(z \mid y=1)p(y=1)}{p(z)} = \frac{K_1}{K}$$

Using Bayes decision rule we will chose the class with the highest probability, which in this case is the class with the highest number of samples in K

Quick summary of KNN

The algorithm can be summarized as:

1. A positive integer k is specified, along with a new sample
2. We select the k entries in our database which are closest to the new sample
3. We find the most common classification of these entries
4. This is the classification we give to the new sample

A few other features of KNN:

1. KNN stores the entire training dataset which it uses as its representation.
2. KNN does not learn any model.
3. KNN makes predictions just-in-time by calculating the similarity between an input sample and each training instance.


Thank You!