JYOTHISHMATHI INSTITUTE OF TECHNOLOGY AND SCIENCE
DEPARTMENT OF COMPUTER SCIENCE ENGINEEING

MACHINE LEARNING

K-nearest neighbor methods

CH.SRINIVAS
ASSOCIATE PROFESSOR

Basic k-nearest neighbor classification

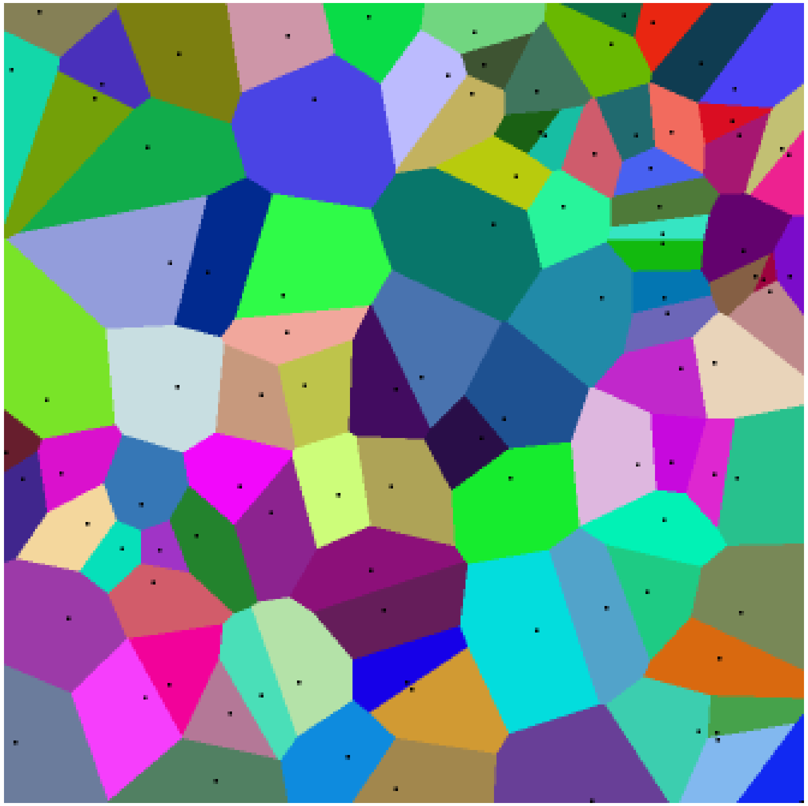
- Training method:
  - Save the training examples
- At prediction time:
  - Find the k training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example x
  - Predict the most frequent class among those y_i's.

- Example:

  http://cgm.cs.mcgill.ca/~soss/cs644/projects/simard/

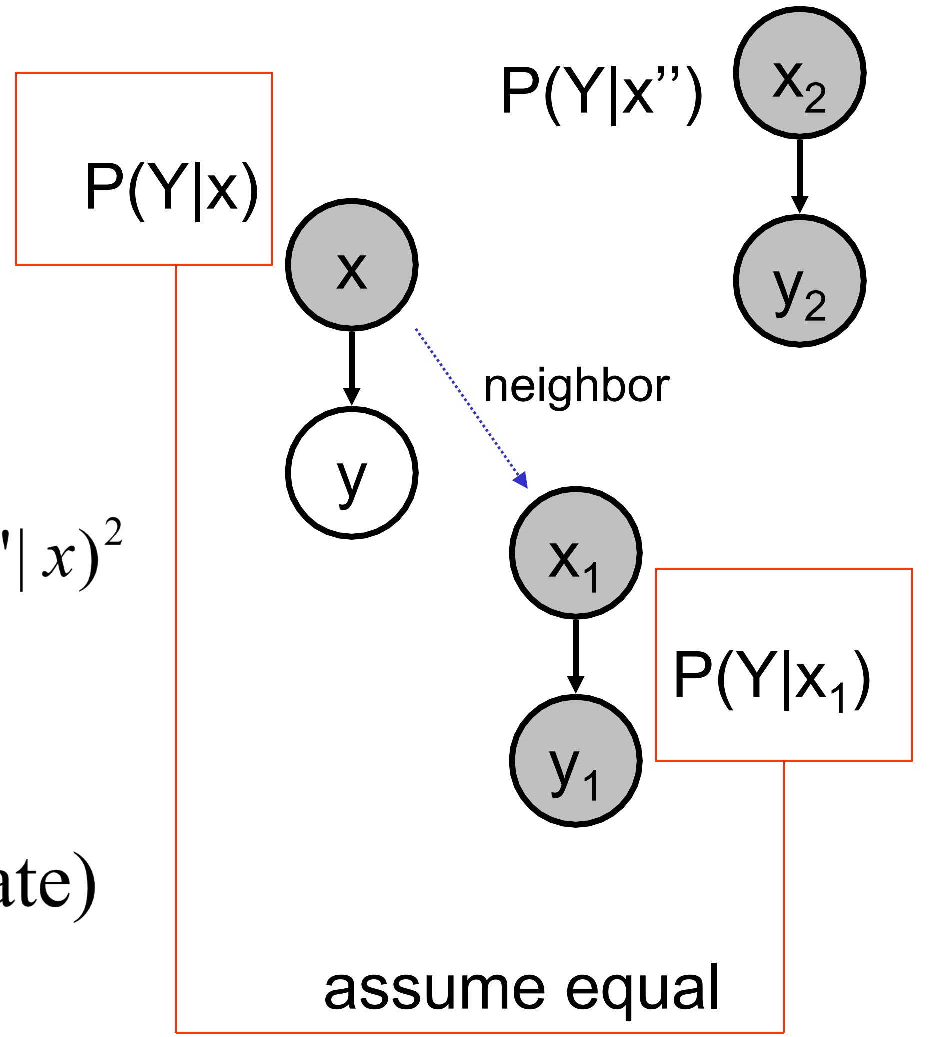
What is the decision boundary?

Voronoi diagram



Convergence of 1-NN

$$
\begin{aligned}
&P(\text{knnError}) \\
&= 1 - \Pr(y = y_1) \\
&= 1 - \sum_{y'} \Pr(Y = y' \mid x)^2 \\
&= 1 - \Pr(y^* \mid x)^2 - \sum_{y' \neq y^*} \Pr(Y = y' \mid x)^2 \\
&\ldots \\
&\leq 2(1 - \Pr(y^* \mid x)) \\
&= 2(\text{Bayes optimal error rate})
\end{aligned}
$$



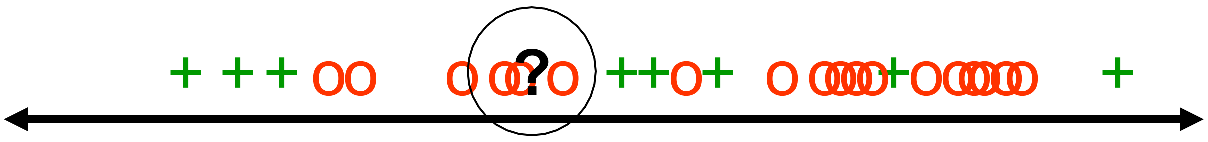
let *y*=argmax Pr(y|x)*

Basic k-nearest neighbor classification

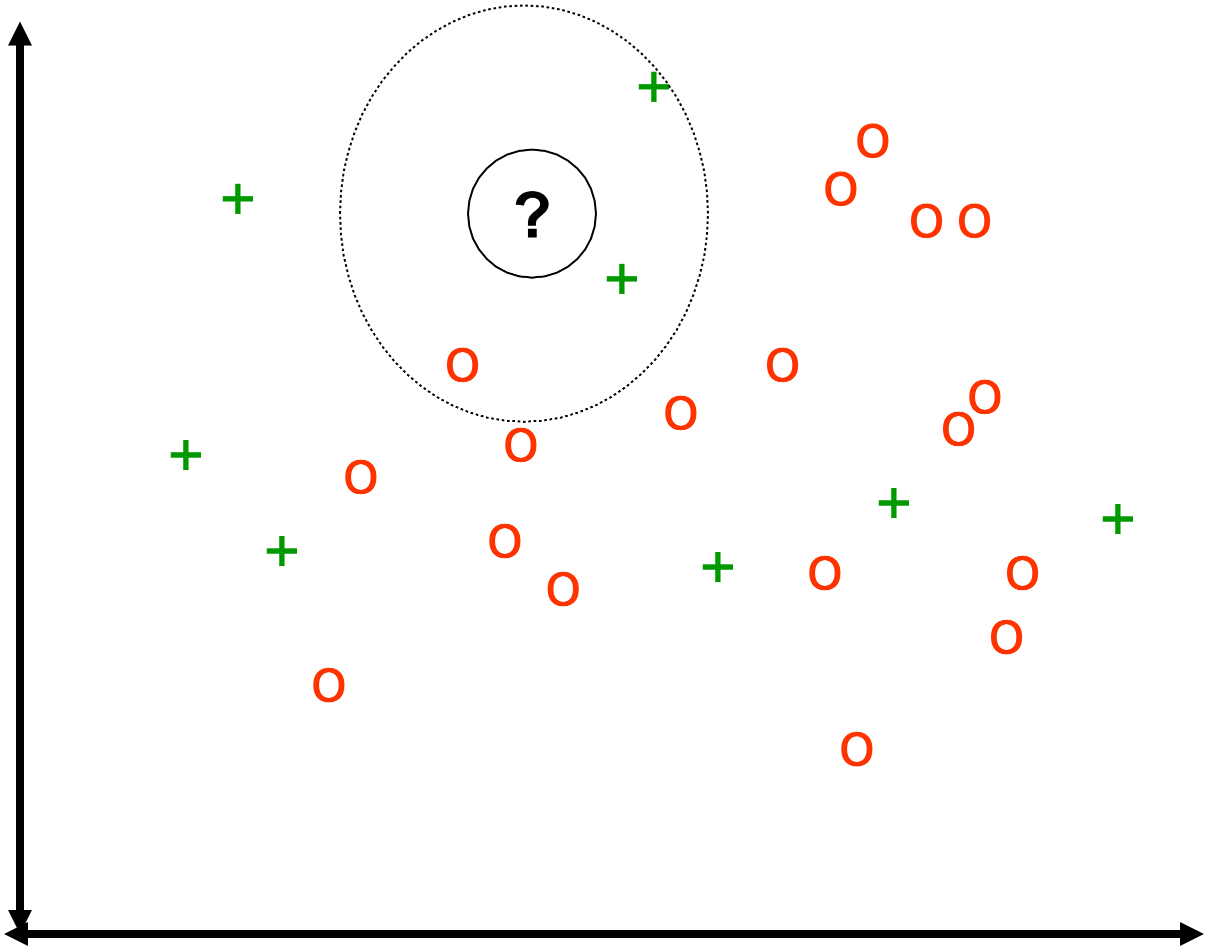
- Training method:
  - Save the training examples
- At prediction time:
  - Find the k training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example x
  - Predict the most frequent class among those y_i's.

- Improvements:
  - *Weighting* examples from the neighborhood
  - Measuring "*closeness*"
  - Finding "close" examples in a large training set *quickly*

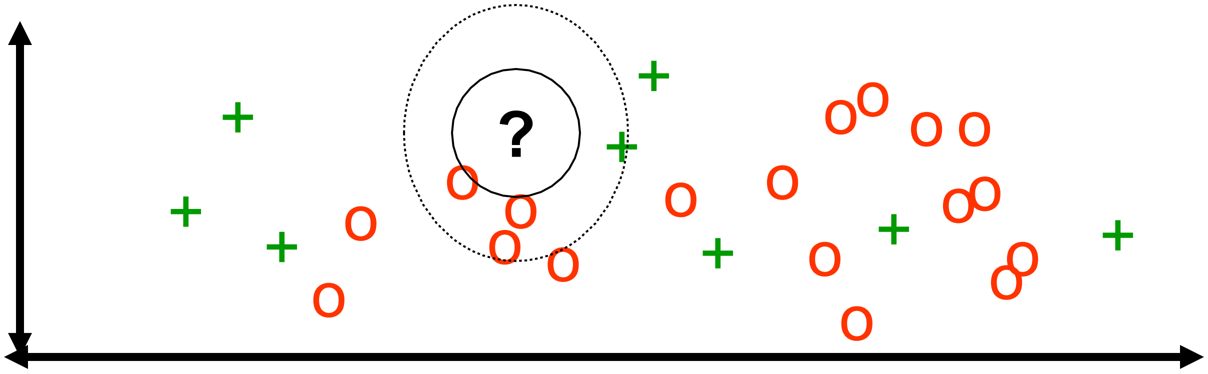
K-NN and irrelevant features



K-NN and irrelevant features



K-NN and irrelevant features



Ways of rescaling for KNN

Normalized L1 distance:

$$\Delta(X,Y) = \sum_{i=1}^{n} \delta(x_i, y_i)|$$

where:

$$\delta(x_i, y_i) = \begin{cases} abs(\frac{x_i - y_i}{max_i - min_i}) & \text{if numeric, else} \\ 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

Scale by IG:

$$\Delta(X,Y) = \sum_{i=1}^{n} w_i \, \delta(x_i, y_i)$$

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$$

Modified value distance metric:

$$\delta(v_1, v_2) = \sum_{i=1}^{n} |P(C_i|v_1) - P(C_i|v_2)|$$

Ways of rescaling for KNN

Dot product:

$$\Delta(X,Y) = dot_{max} - \sum_{i=1}^{n} w_i x_i y_i$$

Cosine distance:

$$\Delta(X,Y) = cos_{max} - \frac{\sum_{i=1}^{n} w_i x_i y_i}{\sqrt{\sum_{i=1}^{n} w_i x_i^2 \sum_{i=1}^{n} w_i y_i^2}}$$

TFIDF weights for text: for doc j, feature i: $x_i = tf_{i,j} * idf_i$:

#occur. of term i in doc j

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}} \qquad \mathrm{idf_i} = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

#docs in corpus

#docs in corpus that contain term i

Combining distances to neighbors

Standard KNN:

$$\hat{y} = \arg\max_y C(y, Neighbors(x))$$

$$\boxed{C(y, D') \equiv |\{(x', y') \in D' : y' = y\}|}$$

Distance-weighted KNN:

$$\hat{y} = \arg\max_y C(y, Neighbors(x))$$

$$\boxed{C(y, D') \equiv \sum_{\{(x',y') \in D' : y' = y\}} (SIM(x, x'))}$$

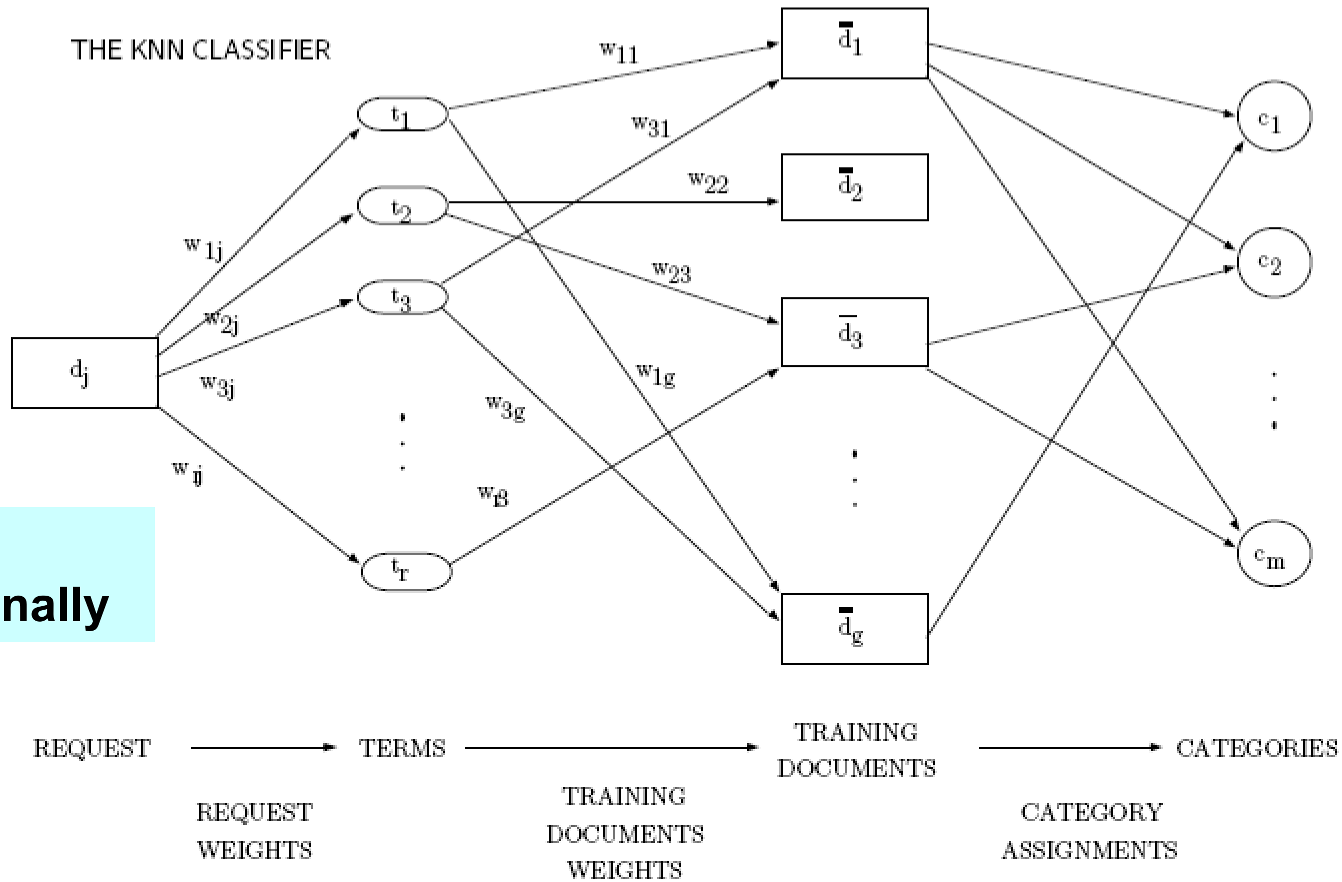
$$\boxed{C(y, D') \equiv 1 - \prod_{\{(x',y') \in D' : y' = y\}} (1 - SIM(x, x'))}$$

$$SIM(x, x') \equiv 1 - \Delta(x, x')$$

Computing KNN: pros and cons

- Storage: all training examples are saved in memory
  - A decision tree or linear classifier is much smaller
- Time: to classify *x*, you need to loop over <u>all</u> training examples *(x',y')* to compute distance between *x* and *x'.*
  - However, you get predictions for every class *y*
    - KNN is nice when there are many many classes
  - Actually, there are some tricks to speed this up…especially when data is sparse (e.g., text)

Efficiently implementing KNN (for text)



IDF is nice computationally

Fig. 5. A graphical representation of the k-NN method. Node d_j has weight equal to 1. Weights flow from left to right and get multiplied by the weights of the edges through which they flow; weights incoming into the same node are summed together. The weight that node c_i receives as a result of the process is the value of $CSV_i(d_j)$.

Tricks with fast KNN

K-means using r-NN

1. Pick k points $c_1=x_1, \ldots, c_k=x_k$ as centers
2. For each x_i, find D_i=Neighborhood(x_i)
3. For each x_i, let c_i=mean(D_i)
4. Go to step 2….

THANK YOU