IAML Past Paper 2015/2016

1. K-Nearest Neighbors

Apply KNN to classify tweets to politics / not politics

a.i. Tweet representation

Represent tweets as a 'BAG OF WORDS'

create vector, each column = words row = documents

attributes: unique words

remove stop words or assign TF-IDF weights.

	A	B	C	D	E
Doc 1	1	0	6	9	10
Doc 2	2	5	7	0	3

$$dist(Doc1, Doc2) = \sqrt{(1-2)^2 + (0-5)^2 + \dots + (10-3)^2}$$

ii. KNN Algorithm

measure distance with all docs

e.g. $K = 3$

$dist(\underline{x}, Doc1) = 1$
$dist(\underline{x}, Doc2) = 3$
$dist(\underline{x}, Doc3) = 2$ → nearest neighbors, check the NNs labels
$dist(\underline{x}, Doc4) = 5$

iii. Decide K

cross-validation, run on multiple Ks, check which one's best

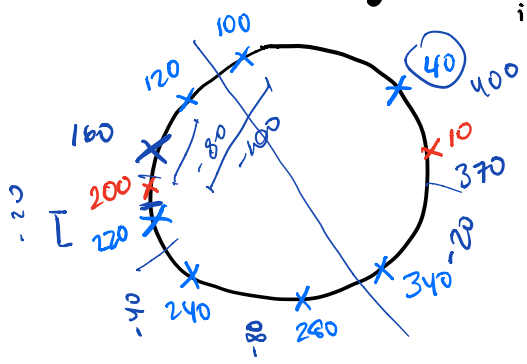
iv. Distance function

$$d(D_1, D_2) = \sqrt{\sum_{i=1,\ x \in D_1,\ y \in D_2}^{n} (x_i - y_i)^2}$$

v. Using K-D Trees?

NO, because K-D trees are for low-dimensional real valued data and we have discrete data.

2. K-means clustering



i. Describe how K-Means can be used

- Place centroids
- repeat until convergence
  - for each pt. x_i: find nearest centroid
  - assign point i to cluster j
  - j -- new centroid = mean of all pts. x_i
- stop when j does not change

ii. K-means

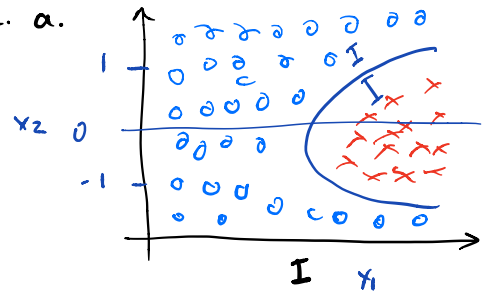
centre = 10: $\frac{40, 340, 100}{3} = 160$ 40 $\quad \frac{340 + 400 + 460}{3} = 400 \approx 40$

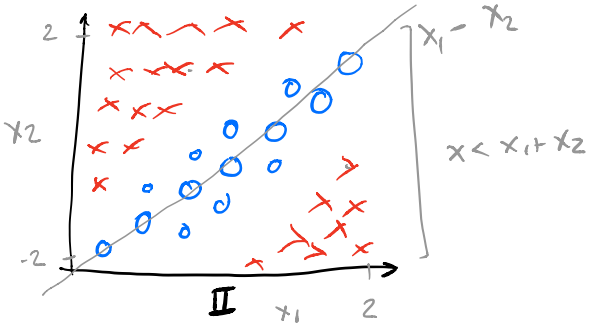
centre = 200 $\quad \frac{120 + 220 + 240 + 280}{4} = 215$

iii. How would you use K-means for hierarchy
- get clusters from each data
- repeat kmeans on each cluster

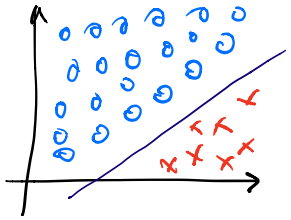
WHAT IS THE DIFFERENCE?

2. a.

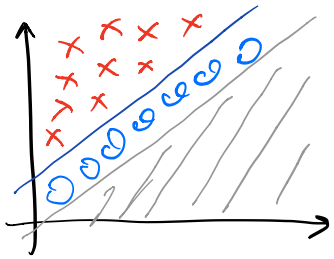




i. For I → square x_2, and then draw a line



ii. For II → $(x_1, x_2) \rightarrow (x_2, x_1)$ for all $x < x_1 - x_2$



b. PCA →

$x_1 = [1\ 2\ 3\ 4]$
$x_2 = [5\ 6\ 7\ 8]$
$x_3 = [9\ 0\ 1\ 2]$
$x_4 = [1\ 6\ 9\ 9]$
$x_5 = [1\ 7\ 9\ 9]$

i. Describe how PCA can be used to transform the matrix X into a lower-dimensional equivalent Z.

1. Center the points → subtract column means.
2. compute covariance matrix

$$cov(h, v) = \frac{1}{n} \sum_{i=1}^{n} h_i v_i$$

3. eigenvectors and eigenvalues
4. pick $m < d$ eigenvectors w/ highest eigenvalues
5. project data points to those eigenvectors

$$b_j (x' - \bar{x})^T e_j \text{ for } j \dots m$$

ii. eigenvectors → $e_1 = [0.00 \quad 0.35 \quad 0.44 \quad -0.82]$ $\lambda_1 = 0.02$

$e_2 = [0.40 \quad -0.76 \quad 0.64 \quad 0.02]$ $\lambda_2 = 0.20$

$e_3 = [0.89 \quad 0.26 \quad 0.25 \quad 0.26]$ $\lambda_3 = 6.44$

$e_4 = [-0.44 \quad 0.47 \quad 0.57 \quad 0.50]$ $\lambda_4 = 38.83$

coordinates of x_1 on e_1 and e_2

?

$$\underset{1\times 4}{[1 \; 2 \; 3 \; 4]} \underset{4\times 2}{\begin{bmatrix} 0.00 & 0.40 \\ 0.35 & -0.76 \\ 0.44 & 0.64 \\ -0.82 & 0.02 \end{bmatrix}} = [x, y] \rightarrow [-1.26 \quad 0.88]$$

$x = 2 \times 0.35 + 3 \times 0.44 + 4 \times -0.82 \approx -1.26$

$y = 0.4 + 2 \times -0.76 + 3 \times 0.64 + 4 \times 0.02 \approx 0.88$

iii. How to select dimensions in PCA?

sort eigenvectors such that $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$.

pick first m eigenvectors which explain 95% of the total variance

↑ eigenvalue λ_i = variance along e_i

i.e.

38.83, 6.44, 0.20, 0.02 total = 45.49

cum. var 38.83 45.27, 45.47, 45.49 95% = 43.21

↳ 2 dimensions

3. Generalization and over-fitting

Linear Regression

Goal = predict a real valued output y for instance vector $\underline{x}$ via a weight vector $\underline{w}$.

Given = training set (x_i, y_i)

Objective Function: Mean Squared Error

i. Difference between generalization error and training error

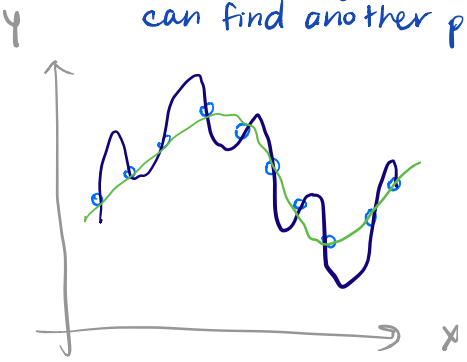
- * gen error → error we will make on future data
- + training error → $E_{train} = \frac{1}{n} \sum_{i=1}^{n} \text{error}(f_D(x_i), y_i)$
- → $E_{gen} = \int \text{error}(f_D[\underline{x}], y)\, p(y, x)\, dx$

ii. Equation for generalization error

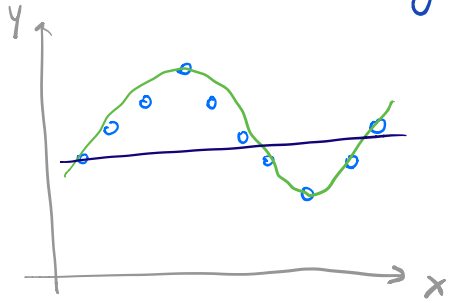
$$E_{gen} = \int \underbrace{\text{error}(f_D(\underline{x}), y)}_{\text{error as before}} \underbrace{p(y, x)}_{\text{how often we see such } x \text{ and } y} dx)$$

iii. Overfitting and underfitting in this case

makes more mistakes on training data but fewer on unseen data. can find another predictor f' w/ smaller E_{tr} & E_{gen}



overfitting



underfitting

b. Evaluation and ROC curves

$TPR = \frac{\#\text{correct}(+)}{\#\text{correct}(+\ \&\ -)}$ $FPR = \frac{\#(-)\text{ incorrect}}{\#(-)\text{ total}}$ $ACC = \frac{\#\text{correct}}{\#\text{total}}$

?

i. simple strategy w/ high accuracy

$$ACC = \frac{TP + TN}{TP + TN + FP + NN}$$

go with the majority of the training data labels (the one w/ highest prior). TPR high, FPR = 0

IAML 2016 with solutions

1. K-Nearest Neighbors

i. 'Tweets' representation
- use a 'Bag of words' representation
- the tweets will be represented as binary (or integer-valued) vectors over the vocabulary, where the vocab = all words in training data
- there will be a separate numeric att. for every word in vocab.
  - if the word is absent in a tweet = 0
    present in a tweet = 1
    * or frequency if using integer-value

ii. Describe KNN algorithm
- computes a distance acc. to a 'distance function' bet. new tweet and each training tweet, and assigns that distance to the training tweet.
- Then, it takes K trn. instances which have the smallest distance and counts whether more of them are labeled as 'politics' or 'not politics'
- tweet label = most frequent label
- if both labels equally frequent, do tie-breaker
  - random coin toss
  - most frequent label in dataset
  - use 1NN.

iii. What distance function would you use?
Euclidean distance.

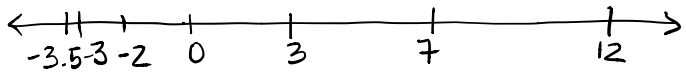
$$D(\underline{x}, \underline{y}) = \sum_{v \in V} (x_v - y_v)^2$$

for tweets $\underline{x}$ and $\underline{y}$, where V is the vocabulary.

iv. Selecting the optimal value of K.
- held-out validation dataset: pick the value of K that gives the lowest classification error over the validation dataset.

v. Can we use K-D trees to speed up THIS KNN?
- no
- dimensionality of tweets is too high (size of vocab)
- inverted indexes are a more appropriate technique

2. Hierarchical Agglomerative Clustering



i. Describe how agglomerative clustering can be used to arrange an arbitrary set of datapoints.

1. Start by assigning each instance to its own cluster
2. Iteratively merge pairs of clusters into a replacement cluster until we are left with a single cluster containing all the instances.
3. This single cluster is the root of the hierarchy (called a dendogram), the two clusters that are merged into it are its children, and so on recursively.

* The algorithm runs for $n-1$ iterations where n is the no. of datapoints.
At each iteration we merge the two clusters that have the smallest distance between them, according to some distance metric.

ii. Single link - defines the distance between two closest elements of the two clusters C_1 and C_2 as the dist. between two closest elements of the two clusters.

How about complete link and average link?

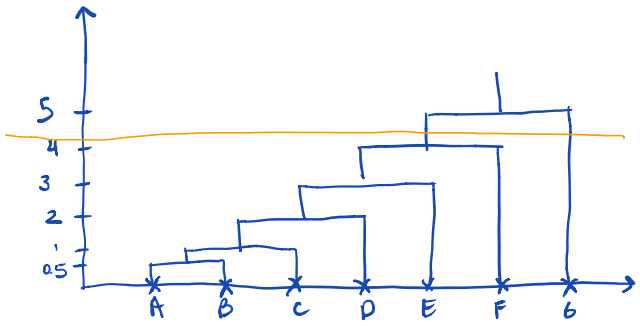
Complete link - distance between two clusters is the distance between the farthest elements in each cluster,
i.e. $\max_{x_1 \in C_1,\, x_2 \in C_2} D(x_1, x_2)$
↓
e.g. Euclidean distance

Average link - mean of all pairwise distances between pairs of points where one point is on one cluster and the other point in the other.

$$\frac{1}{n_1 n_2} \sum_{x_1 \in C_1,\, x_2 \in C_2} D(x_1, x_2)$$

iii. Run single link on dataset and completelink.

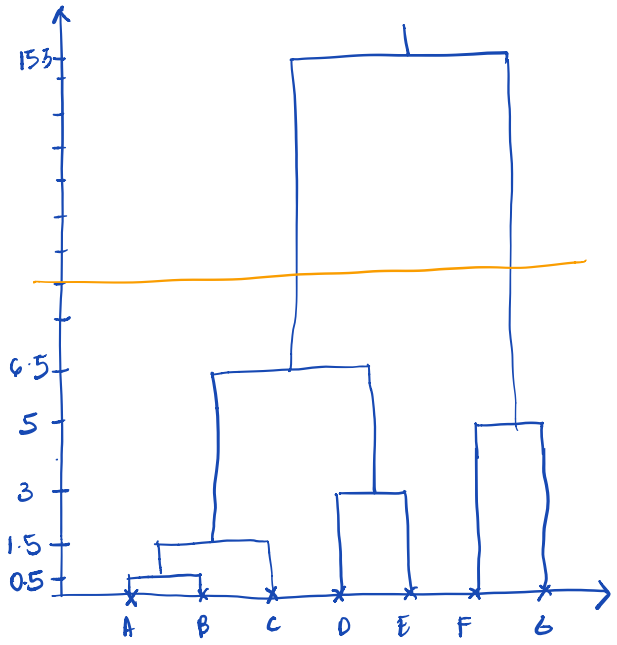
Single link



$C_1 = \{A, B, C, D, E, F\}$

$C_2 = G$

complete link



$C_1 = \{A, B, C, D, E\}$

$C_2 = \{F, G\}$

* Single link has a tendency to cluster instances to long "chains", adding elements one-at-a-time to a large cluster.

* Complete-link → "spherord" clusters, where the diameter of each cluster (dist. bet. a dissimilar points in the cluster) is comparable among all clusters.
  → more balanced: join all singletons to many small clusters, then proceeding to merge them to larger ones.