MLRD Revision Notes

Kevalee Shah

May 20, 2019

Contents

1 Hidden Markov Models

1.1 Definitions

- Markov Assumption - (first order) transitions only depend on the current state:
$$P(w_t|w_{t-1}, \cdots, w_1) \approx P(w_t|w_{t-1})$$
- Markov Chain - stochastic process that embodies Markov Assumption. A probabilistic Finite State Automaton. The states are observable and finite, and transitions are labelled with probabilities. Current situation depends on what happened in the past
- Hidden Markov Model - a markov model in which the system being modeled is a Markov Process with hidden states. We only have access to the observed emission at the given timestep. For each observation there is a number of hidden states, but the association of states and observation is determined by probabilities. The hidden state sequence the corresponds to the observed sequence can only be infered
- States - These are the hidden states. There are two additional states, the start and finish state
- Observation - Output alphabet of observations. Two additional observations, the start and finish symbol.
- Output Independence - probability of an ouput observation depends only on the current state. Not on any other states or observations
$$P(O_t|X_1, \cdots, X_T, O_1, \cdots, O_T) \approx P(O_t|X_t)$$
- Transition Matrix - rows and columns both have the hidden states. Each cell is the probability of moving from state $i \to j$
$$a_{ij} = P(X_t = s_j|X_{t-1} = s_i) \ \frac{count(s_i \to s_j)}{count(s_i \to \text{any state})}$$

  The sum of all transitions into a given state is 1
$$\forall_i \sum_{j=0}^{N+1} a_{ij} = 1$$

  Size = $(N + 2) \times (N + 2)$

  Transitions into start state and out of end state are undefined.

- Emission Matrix - rows have observations and columns have hidden states.

  $b_i(k_j)$ is the probability of emitting vocab item k_j from state s_i
$$b_i(k_j) = P(O_t = k_j|X_t = s_j) \ \frac{count(O_t = k_j \text{ and } X_t = s_i)}{count(X_t = s_i \text{ with any observation })}$$

Size = $(M+2) \times (N+2)$

Undefined for:

- $b_0(k_i)$ - all i except 0
- $b_f(k_i)$ - all i except 0
- $b_i(k_0)$ - all i except 0
- $b_i(k_0)$ - all i except 0
- Therefore only values defined in the first row, first column, last row and last column are $b_0(k_0), b_f(k_f)$

- Dont forget add one smoothing for the above calculations
- Examples of hidden states - speech recognition, speech tagging, machine translation

1.2 Viterbi

We want to write a decoder, so that given a HMM and a sequence O of observations, we get the most likely hidden state sequence that would give us that O. If we were to brute force this, then the search space would be $\mathcal{O}(N^T)$. This is too large and therefore we need to rely on dynamic programming.
There is an optimal substructure property, such that an optimal sequence $X_1 \cdots X_j \cdots X_t$ contains $X_1 \ldots X_j$ which is also optimal and that for $u > t$ then the calculation for the probability of being in state X_t is part of the calculation needed for X_u
Memoising the probability of reaching each state for each time step reduces the search space to $\mathcal{O}(N^2T)$

1.2.1 Data Structure

- Store in a trellis
- Size = $(N+2) \times (T+2)$ - rows are hidden states, and columns are time steps
- For the first row, it is just the probability of state start, with the emission start. $b_0(k_0)$
- Each cell j, t records the probability of the most likely path that ends in state s_j at time t

  $$\delta_j(t) = \max_{1 \le i \le N} [\delta_i(t-1) a_{ij} b_j(O_t)]$$

  Best way to get to s_j from any of the previous states, given the observation of the current timestep
- We also store a helper variable $\psi_j(t)$ which helps us with backtracking. It stores the $(t-1)$ state which gave the highest probability for $\delta_j(t)$

1.2.2 Analysis

- Often averaging isnt useful as the interesting and non interesting instances are both averaged
- Therefore we use precision and recall
- Precision:
$$\frac{\text{Number of correctly predicted X}}{\text{Number of predicted X}}$$
- Recall:
$$\frac{\text{Number of correctly predicted X}}{\text{True number of X}}$$
- F-One Measure - takes equal account of them
$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2 Graphs

2.1 Definitions

- Distance - shortest path between two nodes
- Diameter - longest shortest path between any two nodes in the graph
- Weak/ Strong Ties - closeness of the link
- Giant component - connected component with the most nodes in a graph
- Bridge - An edge, if removed would leave disconnected components
- Local Bridge - edge joining two nodes that have no other neighbours in common. Removing a local bridge icnreases the length of the shortest path
- Triadic Closure - A dynamic property - if A is related to B and A is related to C, it is likely that B is related to C
- Clustering Coefficient - measure of the triadic closure in a graph. The probability that any two neighbours of A are related.
- Gatekeeper Nodes - Let X be a gatekeeper node. That means for some other two nodes, Y and Z, every path connecting the two nodes passes through X. These nodes connect densely populated connected areas of the graph
- Node Betweeness Centrality - The betweeness centrality of a node, say V is the proportion of shortest paths that go through V for each pair of nodes in the graph
- Edge Betweeness Centrality - The betweeness centrality of an edge, say e is the proportion of shortest paths that go through e for each pair of nodes in the graph.

- Clustering - automatically grouping data according to some notion of closeness or similarity
- Agglomerative Clustering - bottom up clustering. Each node starts off in its own cluster and pairs of clusters are merged as one moves up the hierarchy
- Divisive Clutersting - top down clustering by splitting. All nodes start off in one cluster and splits are performed recursively as one moves down the hierarchy

  Newman-Girvan is a method for this type of clustering

  Critereon to break links is edge betweeness centrality
- Classification - assigning data items to predefined classes. Different from clustering as in clustering groupings can emerge unsupervised.
- k - means - A method of cluster analysis, where n observations are partitioned into k clusters in which each observation belongs to the cluster with the nearest mean.
- Hard Clustering - non-overlapping clusters. Each observation has to belong to only one cluster
- Soft Clustering - a single observation can belong to multiple clusters

2.2 Node Betweeness Centraility Equations

- $\sigma(s,t)$: number of shortest paths between s and t

  if $s = t$ then $\sigma(s,t) = 1$
- To calculate $\sigma(s,t)$ recursively:

$$\sigma(s,t) = \sum_{u \in Pred[t]} \sigma(s,u)$$

- $Pred[t] = \{u : (u,t) \in E, d(s,t) = d(s,u) + 1\}$

  In other words, $Pred[t]$ are the nodes one level up from t
- Time complexity to calculate sigmas for all nodes is $\mathcal{O}(V(V+E))$
- $\sigma(s,t|v)$: number of shortest paths between s and t passing through v subitem if $v \in s,t$ then $\sigma(s,t|v) = 0$
- Pairwise Dependencies: For every pair of nodes s,t, and every node v in the graph

$$\delta(s,t|v) = \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

- One-sided dependencies:

$$\delta(s,v) = \sum_{t \in V} \delta(s,t|v)$$

- The Brandes algorithm calculates the above as:

$$\delta(s,v) = \sum_{\substack{(v,w \in E) \\ w:d(s,w)=d(s,v)+1}} \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta(s|w))$$

- $C_B(v)$: the betweeness centrality of v

$$C_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

- Using the calculation of the Brandes algorithm:

$$C_B(v) = \sum_{s \in V} \delta(s,v)$$

2.3 Algorithms

2.3.1 Diameter

```
def getDiameter(Graph g):
  diameter = 0
  for v in g.vertices:
    nodeMax = max(g, v)
    if nodeMax > diameter:
      diameter = nodeMax
  return diameter

def max(Graph g, Node s):
  for v in g.vertices:
    v.distance = 0
    v.visited = false

  queue = Queue([s])
  s.visited = true
  # calculate distance from source to every node using dfs
  while queue != empty:
    current = queue.poll
    currentDist = current.distance
    for w in current.neighbours:
      if w.visited = false:
        queue.add(w)
        w.distance = currentDist + 1
        w.visited = true

  # return max distance
  max = 0
  for v in g.vertices:
    if v.distnace > max:
      max = v.distance

  return max
```

2.3.2 Brandes Algorithm - Node Betweeness

- **input:** directed graph $G = (V, E)$
- **data:** Queue Q, Stack S

  For all $v \in V$

  $dist[v]$: distance from the source

  $Pred[v]$: list of predecessors on the shortest path from source

  $\sigma[v]$: number of shortest paths from source to v

  $\delta[v]$: dependency of source on v
- **output:** betweeness centrality $CB[v]$ initialised to 0

```
for s ∈ V:
  # initialise
  for w ∈ V:
    Pred[w] =  empty list
    dist[W] = ∞
    σ[W] = 0

  dist[s] = 0
  σ[s] = 1
  Q.enqueue(s)

  # calculate sigma
  while Q != empty:
    v = Q.dequeue
    S.push(v)
    for w such that (v,w) ∈ E:
      if dist[w] = ∞:
        dist[w] = dist[v] + 1
        Q.enqueue(w)
      if dist[w] = dist[v] + 1:
        σ[w] = σ[v] + σ[w]
        Pred[w].append(v)

  # calculate delta
  for v ∈ V:
    delta[v] = 0
  while S != empty:
    w = S.pop()
    for v ∈ Pred[w]:
      δ[v] = δ[w] + σ[v]/σ[w] · (1 + δ[w])
    if w ≠ s:
      CB[w] = CB[w] + δ[w]
```

Note: for undirected graphs each pair has been considered twice and therefore the CB values have to be halved at the end.

2.3.3 Brandes Algorithm - Edge Betweeness

Similar to above, with small changes in the backtracking

- **input:** directed graph $G = (V, E)$
- **data:** Queue Q, Stack S

  For all $v \in V$

  $dist[v]$: distance from the source

  $Pred[v]$: list of predecessors on the shortest path from source

  $\sigma[v]$: number of shortest paths from source to v

  $\delta[v]$: dependency of source on v
- **output:** betweeness centrality $CB[q]$ for $q \in V \cup E$, initialised to 0

```
for s ∈ V:
  # initialise
  for w ∈ V:
    Pred[w] =  empty list
    dist[W] = ∞
    σ[W] = 0

  dist[s] = 0
  σ[s] = 1
  Q.enqueue(s)

  # calculate sigma
  while Q != empty:
    v = Q.dequeue
    S.push(v)
    for w such that (v, w) ∈ E:
      if dist[w] = ∞:
        dist[w] = dist[v] + 1
        Q.enqueue(w)
      if dist[w] = dist[v] + 1:
        σ[w] = σ[v] + σ[w]
        Pred[w].append(v)

  # calculate delta
  for v ∈ V:
    delta[v] = 0
  while S != empty:
    w = S.pop()
    for v ∈ Pred[w]:
      c = σ[v]/σ[w] · (1 + δ[w])
      CB[(v, w)] = CB[(v, w)] + c
      δ[v] = δ[v] + c
```

2.3.4 Newman-Girvan Methods

```
while number(connected subgraphs) < specified number:
  calculate edgebetweeness for every edge in graph
  remove edges with highest betweeness
  recalculate number of connected components
```

Ties: either remove all or chose one randomly

3 Sentiment Classification

3.1 Definitions

- Sentiment Classification - task of automatically deciding whether a review is positive or negative based on the text of the review
- Sentiment Lexicon - list of a large collection of words with some associated information, in this case the polarity of the word
- Accuracy - $A = \frac{c}{c+i}$ (where c means correct and i means incorrect)
- Tokenizer - tokenizes the text, which means a sequence of strings in broken into pieces such as words, keywords, phrases and symbols. Punctuation is removed and it has to deal with things such a capital letters etc.
- Machine learning - program that learns from data, adapts after being exposed to new data, implicitly learns from data
- Feature - easily observable properties of the data (e.g. words of review)
- Classes - meaningful labels associated with data (e.g. POS and NEG)
- Classification - function mapping features to classes
- Bayes Theorem -
$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

3.2 Sentiment Lexicon Method

A very simple approach where a counter is used to determine if the review has moree positive or more negative words.
Disadvantages:

- Relies on a fixed set of words
- Humans need to spend time classifying it
- Sentiments of words change over time (e.g. sick)
- This isn't implicit learning, therefore not very scalable

3.3 Probabistic Classifiers

- Features - $O = \{w_1, w_2, \ldots, w_n\}$
- Classes - $C = \{POS, NEG\}$

- A probabilistic cllassfier gives the probability of each class for a given set of features.

$$P(c_i|O) \quad \forall \quad c_i \in C$$

- Therefore for each review we want to calculate $P\{POS|w_1, w_2, \ldots, w_n\}$ and $P\{NEG|w_1, w_2, \ldots, w_n\}$
- The sentiment of the review is then

$$\underset{c \in C}{argmax} P(c|O)$$

3.4 Naive Bayes

Classifier that uses Bayes Theorem $\left(P(c|O) = \frac{P(c)P(O|c)}{P(O)}\right)$ to calculate $P(c|O)$

$$\begin{aligned} c_{NB} &= \underset{c \in C}{argmax} P(c|O) \\ &= \underset{c \in C}{argmax} \frac{P(c)P(O|c)}{P(O)} \\ &= \underset{c \in C}{argmax} P(c)P(O|c) \end{aligned}$$

The $P(O)$ can be ignored as it would be same for all classes

Naive Bayes makes a *naive* independence assumption between the observed features

$$\begin{aligned} P(O|c) &= P(w_1, w_2, \ldots, w_n|c) \\ &\approx P(w_1|c) \times \cdots \times P(w_n|c) \end{aligned}$$

This gives the final equatopm for the Naive Bayes Classifier

$$c_{NB} = \underset{c \in C}{argmax} P(c) \Pi_{i=1}^{n} P(w_i|c)$$

3.4.1 Training and Testing

- *Training* - process of making observations about some known data set

  collect information needed to calculate $P(w_i|c)$ and P(c)
- *Testing* - process of applying the knowledge obtained in the training phase

  apply the formula to get C_{NB}
- *Supervised ML* - use information about classes during training, use the classes that come with the data in the training phase
- Never test on the data you trained with

3.4.2 Parameter Estimation

During the training phase $P(w_i|c)$ and $P(c)$ are estimated
Maximum Likelihood Estimation (MLE) is used.

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)} = \frac{\text{number of times } w_i \text{ occurs in class } c}{\text{number of words in the vocab}}$$

$$\hat{P}(c) = \frac{N_c}{V_{rev}} = \frac{\text{Number of reviews of class } c}{\text{Total number of reivews}}$$

In practice we use *logs*:

$$C_{NB} = \underset{c \in C}{argmax} \log P(c) + \sum_{i=1}^{n} \log P(w_i|c)$$

However there are issues when $P(w_i|c) = 0$ (happens when a certain word doesn't appear in one of the classes). Taking a log of 0 is not allowed and therefore the formula breaks.

Smoothing In order to fix the above problem we use *Laplace Add-on smoothing*:

$$\hat{P}(w_i|c) = \frac{count(w_i|c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w_i|c) + 1}{(\sum_{w \in V}(count(w, c)) + |V|}$$

3.5 Statistical Properties

- Small number of very high frequency words
- Large number of low frequenct words
- *Zipf's Law:* the nth most frequent word has a frequency proportational to $\frac{1}{n}$ - word's frequency is inversely proportional to rank
- $$f_w = \frac{k}{{r_w}^{\alpha}}$$

  where:

  f_w - frequency of word

  r_w - rank of word

  α, k - constants that vary with languages ($\alpha = 1$ for english)
- *Type* - any unique word.

  Number of types is the size of the dictionary
- *Token* - instance of a type

  e.g. Moby Dick has 13721 *the* tokens
- As the size of the text increases, the number of new types decreases

- *Heap's Law* - describes the relationship between the size of the vocabularly and size of the text it is from.
- $$u_n = kn^\beta$$

  where:

  u_n - number of types (vocabularly size)

  n - number of tokens (text size)

  β, k - language dependent constants ($\beta \approx \frac{1}{2}$, $30 \leq k \leq 100$)
- Plotting Zipf curves and Heap's Law in log-space and then fitting a line of best fit helps us find the unknown constants

Effect on Classifier When we were using MLE an estimatiton for $P(w_i|c)$, only the *seen* types receive a probability estimate

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

The total probability sums to 1, but that is too big as it doesn't account for the words not seen as the text size of the training data is too small.
The estimated probabilites is too big.

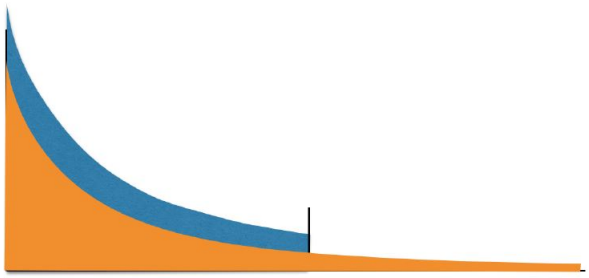


Figure 1: MLE(blue) overestimates probability of seen type

Smoothing therefore helps to redistribute the probability mass - it takes some portion away from the MLE overestimate and redistributes it to the unseen types.

3.6 Significance Testing

All documents vary and therefore some documents will make it easier for one system to perform better than the other system. Therefore it is important to test whether two systems are *significantly different* to each other.

- *Null Hypothesis:* two results come from the same distribution and therefore there is no significant difference between the two systems. Any observed difference is due to sampling/ experimental error.

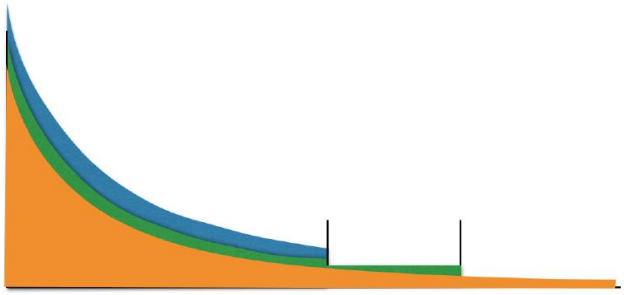


Figure 2: Effect of smoothing

- *Significance Level:* The probability of rejecting a null hypothesis when it is really true (e.g. $\alpha = 0.01$ or $\alpha = 0.05$)
- The null hypothesis can then be rejected with a confidence of $1 - \alpha$
- Rejecting the null hypothesis means that the observed result had a really low probability of occuring due to chance.

3.6.1 Sign Testing

- One variation used a *binary event model*

  event - a review

  binary - each even has a binary outcome

  Positive - System 1 beats Sytem 2 for this event

  Negative - System 2 beats System 1 for this event

  Tie - read further on

- The binomial distribution is used to calculate the probability that at least a certain number (say k)of events are positive/ the probability that at most $total - k$ number of events
- $$P(X \leq k|N) = \sum_{i=0}^{k} q^i (1-q)^{N-i}$$

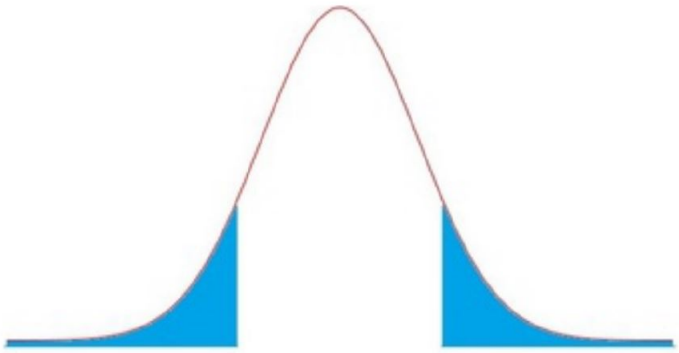
  where:

  $P(X \leq k|N)$ means the probability of observing at most k negative events out of N events

  $q = 0.5$

- If that probability we calculate is less than our significance level then there is less that $signifincance\%$ that System 2 beats System 1 and therefore the null hypothesis can be rejected.
- This can be extended to a *two-tailed test* - testinf for statistical significance regardless of direction

- Now we are interested in the value of k for at which the significance probability exists in the two tails, not just one



- Since $q = 0.5$, $B(N, 0.5)$ is symmetrice and therefore the probability we want to calculate is $2P(X \leq k) \leq$ significance probability
- Ties can be split between the positive and negative count
- Overall the following needs to be calculated

$$2\sum_{i=0}^{k} \binom{n}{i} q^i (1-q)^{n-i}$$

where:

$n = 2\lceil \frac{Null}{2} \rceil + Plus + Minus$

$k = \lceil \frac{Null}{2} \rceil + min\{Plus, Minus\}$ (number of cases with the least common sign)

$q = 0.5$ Note:

Dividing null cases evenly and round up

Big number so need to use BigInteger in Java

3.7 Overtraining

A classifier *generalises* well when it is able to perform well on new, unseen data. This means the classifier should be able to recognise characteristics of the data that are general enough to apply to unseen data and ignore the characteristics that are specific to the training dataset.

It is important never to test on training data.

Overtraining - when you think you are making improvements as performance on the test data is increasing, however in reality the classifier is becoming worse as it is less able to generalise.

Type Three Errors - constantly rejecting the null hypothesis but for the wrong reason

Signs of Overtraining

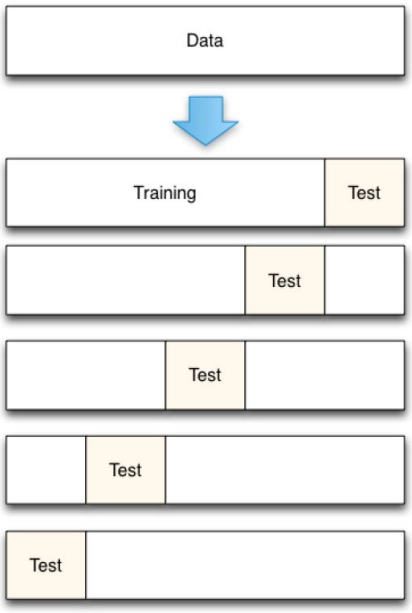
- Using a small amount of data and using the same data every time you make an improvement to the classifier
- If the most characteristic feature for each class are features that are unlikely to generalise
- Time effects - e.g. the Wayne Rooyne effect; when time changes also mean changes in perception to the same feature. E.g. people or words

3.8 Cross Validation

Motivation - it is not possible to get new data every time you make a small improvement, and neither is it allowed to test on the training set. However we also want to use as much of the training data as possible as that directly affects how well the ML system will do. Cross-validation is a way by which every little bit of the training data can be used for testing.

Method

- Split data into N folds
- For each fold, use that fold for testing and the rest for training.
- Average the results for each fold to get the average performance



Stratified Cross Validation - each split is done in such a way that it mirrors the distribution of the classes observed in the overall data. e.g. split so that there

are the same number of positive and negative reviews in each fold

Result For each fold a significance test can be done which means that more usable test data is gained, and are more likely to pass the test.

Variance To see how similarly the splits performed the variance can be calculated

$$var = \frac{1}{n}\sum_{i}^{n}(x_i - u)^2$$

where:

- x_i - score of the i^{th} fold
- u - average of the scores
- n - number of folds

3.9 3-Class Classifier

Most reviews aren't positive or negative, they are neutral. And therefore

3.10 Human Disagreement

It is often hard to determine what the ground truth should be anywway. Human agreement is the only empirically available source of truth in decisions which are influenced by subjective judgements.
Something is 'true' if several humans agree on their judgement, and the more they agree the more true it is.

Pairwise Observed Agreement - the metric used to see how much a group of people agree. The average ratio of observed to possible rater-rater agreements.

$$\bar{P}_a = MEAN\left(\frac{\text{observed pairs in agreement}}{\text{possible pairs}}\right)$$

- There are $\binom{n}{2}$
- E.g. in a group of 5 people, there are 10 possible pairs. If the split in the group was $3, 2$, then the ratio would be

$$\frac{\binom{3}{2} + \binom{2}{2}}{\binom{5}{2}} = \frac{4}{10}$$

Pairwaire Chance Agreement - need a way to see how much better the observed agreement is compared to what we would just get by chance. This is the proportion of rater-rater agreement that we would expect by chance, denoted $\bar{P}_e$
It is calculated as the sum of square of probabilities of each category.

$$\bar{P}_e = \sum_{c \in C} p(c)^2$$

Kappa - This uses the observed and chance agreement to measure the reliability of agreement between a fixed number of raters when assigning categorical ratings. It calculated the degree of agreement over that which would be expected by chance.

$$\kappa = \frac{\bar{P}_a - \bar{P}_e}{1 - \bar{P}_e}$$

- $\kappa = 1$ is total agreement
- $\kappa = 0$ no agreement beyond what we could expect by chance
- $\kappa < 0$ if observed agreement is less that what we'd expect from chance
- κ is affected by the number of categories and also can be misleading in a small sample size
- Therefore no universal agreed interpretation
- To calculate $\bar{P}_e$ and $\bar{P}_a$:

$$\bar{P}_e = \sum_{j=1}^{k} \left(\frac{1}{N} \sum \frac{n_{ij}}{n_i} \right)^2$$

$$\bar{P}_a = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{n_i(n_i - 1)} \sum_{j=1}^{k} n_{ij}(n_{ij} - 1)$$

where:

n_{ij} - number of predictions that item i belongs to class j

$n_i = \sum_{j=1}^{k} n_{ij}$ - total number of predictions for item i