Machine Learning and Real World Data

Sentiment Classification

Is a standard task in NLP.

Definitions
- A **type** is a unique word.
- A **token** is an instance of a type.
- A **tokeniser** split text into tokens.

Naive Bayes Classification

Machine learning is a program that adapts its behaviour after being exposed to new data: without explicit programming, and implicitly from the data alone.

Definitions
- **Features** are observable properties of the data.
- **Classes** are labels associated with the data (e.g. positive/negative)
- **Classification** is a function that maps features to classes.

The classifier is given a set of features O and classes $c_1, \ldots, c_n \in C$, and gives $P(c_i|O)$ for each c_i.
- $O = \{w_1, w_2, \ldots, w_n\}$ are the words in the review.
- $C = \{\text{pos}, \text{neg}\}$

Choose the $P(c_i|O)$ with the highest probability

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

As $P(O)$ is a constant and does not affect argmax.

$$\begin{aligned} P(O|c) &= P(w_1|c) \times P(w_2|C) \times \cdots \times P(w_n|c) \\ c_{NB} &= \underset{c \in C}{\operatorname{argmax}}\ P(c) \prod_{i=1}^{n} P(w_i|c) \\ &= \underset{c \in C}{\operatorname{argmax}}\ \left(\log P(c) + \sum_{i=1}^{n} P(w_i|c) \right) \end{aligned}$$

Summing results in less floating point precision errors than multiplying.

In training, collect information needed to calculate $P(c)$ and $P(w_i|c)$

$$\begin{aligned} P(c) &= \frac{N_c}{N_{\text{total}}} \\ P(w_i|c) &= \frac{\text{count}(w_i \text{ in } c)}{\sum_{w \in V} \text{count}(w \text{ in } c)} \end{aligned}$$

Laplace smoothing prevents $P(w_i|c)$ from being zero by adding 1 to the count of each type.

$$
\begin{aligned}
P(w_i|c) &= \frac{\text{count}(w_i \text{ in } c) + 1}{\sum_{w \in V} (\text{count}(w \text{ in } c) + 1)} \\
&= \frac{\text{count}(w_i \text{ in } c) + 1}{\sum_{w \in V} \text{count}(w \text{ in } c) + |V|}
\end{aligned}
$$

Name	**Description**
Zipf's law	$f_w \approx \frac{k}{(r_w + \beta)^\alpha}$ • f_w is the frequency of the word • r_w is the frequency rank of the word • k, α and β are language dependent constants Note: β is the rank shift.
Heap's law	$u_n = kn^\beta$ • u_n is the number of types (vocabulary size) • n is the number of tokens • β and k are language dependent constants

In Naive Bayes Classification, only seen types receive a probability estimate. Adding 1 redistributes some probability mass mass to unseen types.

Significance Testing

- The **null hypothesis**: the two result sets comes from the same distribution.
- Rejecting the null hypothesis means the observed results is unlikely to have happened by chance.

Choose a **significance level** α, reject the null hypothesis if the probability of observing the event under the null hypothesis is less than α.

In a **binomial distribution** $B(N, q)$

$$
P(X = k) = \binom{N}{k} q^k (1 - q)^{N-k}
$$

$$
P(X \leq k) = \sum_{i=0}^{k} \binom{N}{i} q^i (1 - q)^{N-i}
$$

A **two-tailed test** tests if the two systems performs equally well: the α in each tail is halved.

	Actual = same	**Actual = different**
Predicted = same	Correct	Type II error: • Use a more powerful test (e.g. permutation test rather than sign test) • Use more data
Predicted = different	Type I error	Correct

Significance testing cannot show two distributions are the same.

Note
For testing sentiment classifiers, ignoring ties will lead to the null hypothesis being incorrectly rejected. Add 0.5 to the count of positive and negative results in case of ties.

Overtraining
Overtraining is where more training makes the classifier perform worse on unseen data.

Am I overtraining?
- If you are using large amounts of new test data, not overtraining.
- If incrementally improving the classifier on the same small test data, overtraining.

Overtraining is caused by finding characteristic features of each class that are hard to generalise.

N-fold cross-validation
1. Split data into N equal folds.
2. For each fold X, train on other folds, test on fold X only.
3. Average all the accuracy for the final accuracy.

It is good if each splits performs equally well, calculate the variance:

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

Consider the N experiments as one overall experiment.

Cross validation	Description
Stratified cross-validation	Each split mirrors the distribution of classes in the overall data.
Jack-knitting	Each individual data point is a split.
Dependency-sensitive cross-validation	Fold in a way that known characteristics of a data are isolated (e.g. one split per genre)

Cross-validation does not solve the problem of overtraining.

Instead, a **validation corpus** (a separate set of data not used for training or testing) can be used to
- Tweak parameters before training
- Check if training is making the system perform worse on the validation corpus (is it overtraining?)

Uncertainty and Agreement

$$\overline{P_a} \text{ observed agreement} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{\#\text{observed pairs of agreements on item } i}{\#\text{possible pairs}}$$

where N is the number of items to be classified.

$$\overline{P_e} \text{ chance agreement} = \sum_{c \in C} P(c)^2$$

Fleiss' Kamma

$$\kappa = \frac{\overline{P_a} - \overline{P_e}}{1 - \overline{P_e}}$$

- $\kappa = 1$ then complete agreement
- $\kappa = 0$ then no agreement beyond what is expected by chance
- $\kappa = 0.8$ means very good agreement

Note
κ can be negative.

Social Networks

Definitions
- **Distance** is the length of the shortest path between two nodes.
- The **diameter** of a graph is the maximum distance between any pair of nodes.
- The **degree** of a node is the number of neighbours it has.

Natural networks often have the **small world property**.

Definition
A **small world network** is one with
- Not many connections
- There are closely clustered regions
- Clusters are connected by only a few links

The measurable characteristics are
- **High clustering coefficient**: a node neighbours are likely to be neighbours of each other.

  Definitions
  - **Triadic closure** is if $A \leftrightarrow B$ and $A \leftrightarrow C$, then very likely $B \leftrightarrow C$.
  - **Global clustering coefficient** is

  $$\frac{\#\text{ of closed triads}}{\#\text{ of possible closed triads}}$$

- **Small average path length**, typically grows logarithmically with network size.
- **Sparse connectivity**: has few edges compared to the fully connected graph.

Definitions
- **Giant component** is a connected component containing most of then nodes in a graph.
- **Weak ties** are distant links, oppisite of a **strong link**.

  Note
  Links that keep two giant components together are often weak ties.

- **Bridge** is an edge that connects to components which would otherwise be unconnected.
- **Local bridge** is an edge when removed, the two nodes will have no neighbours in common. (shortest path between the two nodes increases)

Betweeness Centrality

Definition
A **gatekeeper** are cruicial edges in linking densly connected regions in the graph.

Define

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

$$\sigma(s,t) = \sum_{u \in \text{ pred}(t)} \sigma(s,u)$$

  where $\text{pred}(t) = \{u \mid (u,t) \in E \wedge d(s,t) = d(s,u) + 1\}$
- $\sigma(s,t|v)$ number of those paths passing through v

Note

- If $s = t$, then $\sigma(s,t) = 1$
- If $v \in \{s,t\}$, then $\sigma(s,t|v) = 0$

Definitions
Betweeness centrality of v

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

Another way to calculate $C_B(v)$:

Define **pairwise dependencies** $\delta(s,t|v)$ and **one-sided dependencies** $\delta(s|v)$

$$\begin{aligned}
\delta(s,t|v) &= \frac{\sigma(s,t|v)}{\sigma(s,t)} \\
\delta(s|v) &= \sum_{t \in V} \delta(s,t|v) \\
&= \sum_{\substack{(v,w) \in E \\ w : d(s,w) = d(s,v) + 1}} \frac{\sigma(s,v)}{\sigma(s,w)} \cdot (1 + \delta(s|w)) \\
C_B(v) &= \sum_{s \in V} \delta(s|v)
\end{aligned}$$

Algorithm 1: Brandes Algorithm

for $s \in V$ **do**
  ▹ Initialisation
  for $w \in V$ **do**
    $\text{pred}[w] \leftarrow$ empty list
    $\text{dist}[w] \leftarrow \infty$
    $\sigma[w] \leftarrow 0$
    $\delta[w] \leftarrow 0$
  end
  $Q \leftarrow \text{empty list}$
  $Q \leftarrow \text{enqueue } s$
  $S \leftarrow \text{empty stack}$

```
    ▷ Calculate σ for all v ∈ V
    while Q not empty do
        v ← dequeue Q
        S ← push v

        for w : (v, w) ∈ E do
            ▷ Finding w for the first time
            if dist[w] = ∞ then
                dist[w] ← dist[v] + 1
                Q ← enqueue w
            end

            ▷ Update number of shortest paths through w
            if dist[w] = dist[v] + 1 then
                σ[w] ← σ[w] + σ[v]
                pred[w] ← append v
            end
        end

        ▷ calculate C_B[w]
        while S not empty do
            pop w ← S
            for v ∈ pred[w] do
                δ[v] ← δ[v] + σ[v]/σ[w] · (1 + δ[w])
            end
            if w ≠ s then
                C_B[w] ← C_B[w] + δ[w]
            end
        end
    end
end
```

For bidirectional graphs, divide the value of each $C_B[w]$ by 2.