**Natural Language Processing (NLP)**

**Session 2**

1. **Text Classification**:
   1. *Supervised*:
      1. *Input*: a document, a fixed set of classes and a training set of labelled data
      2. *Output*: a learned classifier / model
         1. Generative: build a model by aggregating the features of a class e.g., Naïve Bayes.
         2. Discriminative: build a model by separating the features of a class e.g., logistic regression.
         3. Discriminative vs. Generative:
            1. SVMs and other discriminative classifiers can outperform NB and generative approaches with enough data.
            2. However, SVMs and other more complex approaches are less interpretable than Bayesian approaches.
            3. Logistic Regression can give a bit of the best of both worlds, as its co-efficient (feature weights) can be interpreted, and it can reach high-performance with optimization.
2. **Naïve Bayes**: Based on Bayes Rule:  
     
   P(A given B) = (P(B given A) multiplied with P(A)) / P(B)  
     
   If we make very naive assumptions about the generative model for each label, we can find a rough approximation of the generative model for each class, and then proceed with the Bayesian classification.
   1. Advantages:
      1. Works quickly and can save a lot of time.
      2. Suitable for solving multi-class prediction problems.
      3. If its assumption of the independence of features holds true, it can perform better than other models and requires much fewer training data.
   2. Disadvantages:
      1. Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.
      2. This algorithm faces the ‘zero-frequency problem’ where it assigns zero probability to a categorical variable whose category in the test data set wasn’t available in the training dataset. It would be best if you used a smoothing technique to overcome this issue.
      3. Its estimations can be wrong in some cases, so you shouldn’t take its probability outputs very seriously.
3. ***Maximum a posteriori* (MAP) hypothesis for classification**:

cMAP­ = maximise P(text in document *X* given classes cj) times P(classes cj)

* 1. Density estimation is the problem of estimating the probability distribution for a sample of observations from a problem domain.
  2. MAP estimation is a probabilistic framework for solving the problem of density estimation.
  3. MAP involves calculating a conditional probability of observing the data given a model weighted by a prior probability or belief about the model.
  4. MAP provides an alternate probability framework to maximum likelihood estimation for machine learning.

1. **Naïve-Bayes Classifier**:  
     
   Classifies based on a new instance of a tuple of attribute values. <x­1, x­2, x­3 … x­n>  
     
   cNB= maximise P(tuple of values <x­1, x­2 … x­n> given classes cj) times P(classes cj)  
     
   where,
   1. P(cj): Prior can be estimated from the frequency of classes in the training examples/documents.
   2. P(tuple of values <x­1, x­2 … x­n> given classes cj): simple assumption for text: attributes are text/word positions, values are words.

To avoid sparsity of set of attributes, assume, naively, that:

* Features are unordered words (a “bag of words”).
* Features (words appearing) are conditionally independent, so the probability of observing the conjunction of features is equal to the product of the individual probabilities.
* So, the classifier can scan through the positions in a text, but each position xi is treated as having the same parameters regardless of its position:  
    
  cNB= maximise P(classes cj) times the multiplied product of the set P(position xi given classes cj)  
    
    
    
    
    
    
    
    
    
    
  *Underflow protection*:
  + Multiplying lots of probabilities, which are between 0 and 1, can result in floating-point underflow when calculated by a digital computer with finite memory.
  + *Log(xy) = log(x) + log(y)*, it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities first before obtaining the log.  
      
    cNB= argmaxlog (maximum likelihood) of P(classes cj) + sum of the log of P(tuple of values <x­1, x­2 … x­n> given classes cj).  
      
    Class with highest final un-normalized log probability score is still the most probable.

1. **Smoothing**: Zero probabilities (noise) cannot be conditioned away, however for training purposes, the noise can be problematic i.e., infinity values, etc. Data smoothing uses an algorithm to remove noise from a data set, allowing important patterns to stand out.  
     
   ‘Pretend’ we’ve seen each word co-occur with each class once more than observed in training:  
     
   P(words wi ­given class c) = (count(words wi in class c) + 1) / (sum of count(all words w contained in class c) + size of the vocabulary).
2. **Text Classification Algorithms – Learning:**
   1. Aim: From training corpus C, extract vocabulary V
   2. Calculate required P(class cj) and P(word xk given cj )terms:
      1. For each cj in C do:
         1. docsj = subset of documents for which the target class is cj.  
              
            P(cj) = docsj / total # of docs
         2. Textj = single document containing all docsj**.**
         3. For each word xk in vocabulary V do:
            1. nk = number of occurrences of xk in textj.
            2. Calculate probability of P(xk given cj) that is equal to nk + 1 divided by n + size of the vocabulary
3. **Text Classification Algorithms – Classifying:**
   1. Positions = Locate all word positions in current document which contain tokens found in vocabulary V.
   2. Return cNB.

**Measuring Performance**:

Let's make the following definitions:

* "Wolf" is a **positive class**.
* "No wolf" is a **negative class**.

A true positive tp is an outcome where the model correctly predicts the positive class. Similarly, a true negative tn is an outcome where the model correctly predicts the negative class.

A false positive fp is an outcome where the model incorrectly predicts the positive class. And a false negative fn is an outcome where the model incorrectly predicts the negative class.  
  
A 2 x 2 confusion matrix example:

|  |  |
| --- | --- |
| **True Positive (tP):**  Reality: A wolf threatened.  Shepherd said: "Wolf."  Outcome: Shepherd is a hero. | **False Positive (fp):**  Reality: No wolf threatened.  Shepherd said: "Wolf."  Outcome: Villagers are angry at shepherd for waking them up. |
| **False Negative (fn):**  Reality: A wolf threatened.  Shepherd said: "No wolf."  Outcome: The wolf ate all the sheep. | **True Negative (tn):**  Reality: No wolf threatened.  Shepherd said: "No wolf."  Outcome: Everyone is fine. |

1. **Precision**: Precision is the fraction of true positive examples among the examples that the model classified as positive. In other words, the number of true positives divided by the number of false positives plus true positives.  
     
   Precision = tp / (tp + fp)  
     
   e.g.,  
   Precision = #spam correctly predicted / #spam predicted
2. **Recall**: Recall, also known as sensitivity, is the fraction of examples classified as positive, among the total number of positive examples. In other words, the number of true positives divided by the number of true positives plus false negatives.  
     
   Recall = tp / (tp + fn)  
     
   e.g.,  
   Recall = #spam correctly predicted / #actual spam messages
3. **F-score**: The F-score, also called the F1-score, is a measure of a model’s accuracy on a dataset. It is used to evaluate binary classification systems, which classify examples into ‘positive’ or ‘negative’.  
     
   The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model’s precision and recall.  
     
   F1-score = 2 \* Precision \* Recall / (Precision + Recall)  
     
   The F-score is commonly used for evaluating information retrieval systems such as search engines, and also for many kinds of machine learning models, in particular in natural language processing.
4. **Feature Classification**:
   1. *Tokenisation*: split a string into words
   2. *Normalisation*: have predictable, regular forms
   3. *Stemming*: Porter’s algorithm
   4. *Feature selection*:
      1. Mutual information: In training set, choose *k* words which best discriminate the categories.  
           
         Pointwise mutual information:  
           
         I(x, c) = log of P(x intersect c / P(x) times P(c)  
           
         For each category we build a list of *k* most discriminating terms.
      2. Information gain: Metric much used in ML to decide on the importance of features. Measures how much a given feature (word) reduces the entropy.  
           
         Entropy is the amount of uncertainty in a distribution.

**Discriminative approaches**:

1. **Logistic Regression**: Discriminative nonlinear classifier. Summation of the element-wise product of the weight matrix and the feature vector.   
     
   Logistic Regression = maximise the posterior probability P(A given B)  
     
   • Output is a linear combination of features   
   • i.e. weight matrix w, feature vector x   
   • … with exponential to make it positive ...   
   • … and normalised to [0-1].   
     
   A graph representing this would look like a sigmoid function (slanting s-shape) contained between 0 and 1 on the y-axis.  
     
   P(y given x) = 1 / (1 + exp raised to the power of -(weights dot product with feature vector x))
2. **Linear classifiers**: An alternative to using something like LR and the exponential function, is to keep to linear classification, but to transform the dataset into a dimension in which the classes are linearly separable.  
     
   Separator can be expressed as f(x) = wx + b, which is similar to the equation of a straight-line graph y = mx + c. There are different gradients of the lines that can separate data through linear classification. Choice depends on the application.
3. **Hyperplanes**: To enable linear classification in dimensions higher than 2, need hyperplanes.   
     
   Separator expressed as ax + by = c. Same as linear classifiers, plenty of choices to separate available for one given data set. Support Vector Machine (SVM) finds an optimal solution.
4. **Support Vector Machine (SVM)**:
   1. For an optimum solution to the available hyperplanes, supporting vectors/lines are created around the margin that separates the data points.
   2. SVMs maximize the margin around the separating hyperplane.
      1. The distance between the points and the line are as far as possible.
   3. The decision function is fully specified by a subset of training samples, the support vectors.
   4. Constraint optimisation problem:
      1. Optimisation is maximize the margin.
      2. Constraint is that the data points can’t be on the margins.
   5. *Linear SVM* (LVSM): Implies that only support vectors are important; other training examples are ignorable.  
        
      Maximum margin can be optimised by the following constraints:

The dot product of datapoint xi and the hyperplane normal vector (90deg to the original vector) should be more than positive 1 for data points classed as positive. And vice versa for negative datapoints i.e., xi dot w less than -1.  
  
 Given a new point x, we can score its projection onto the hyperplane normal:   
• i.e., compute score

• Decide class based on whether < 0 or > 0

• Can set confidence threshold t.

1. **Discriminative vs. Generative**:
   1. SVMs and other discriminative classifiers can outperform NB and generative approaches with enough data.
   2. However, SVMs and other more complex approaches are less interpretable than Bayesian approaches.
   3. Logistic Regression can give a bit of the best of both worlds, as its co-efficient (feature weights) can be interpreted, and it can reach high-performance with optimization.
2. **Cross-validation**:
   1. Cross-validation over multiple splits
      1. You can repeat the train/heldout split multiple times by iterating over the training data.
   2. For k-fold cross-validation, break up data into k folds.
      1. (Equal positive and negative inside each fold, or random?)
   3. For each fold
      1. Choose the fold as a temporary test set.
      2. Train on k-1 folds, compute performance on the test fold.
   4. Report average (macro) performance of the k runs (and micro too over all instances if useful).
   5. Better handles sampling errors/biases from different datasets.