**REPORT**
**Statistical Methods In Artificial Intelligence Project**

**Title: Category Ranking**
**Submitted to: Ravi Kiran Sarvadevabhatla**

**Team Members:**
**2018201009: Sarvat Ali**
**2018201053: Prabha Pandey**
**2018201054: C Sai Sukrutha**
**2018202010: Shruti Chandra**

**Assigned TA: Himanshi Sharma**

**Description:**

Topic-ranking of document is where each document can be associated with multiple relevant topics. In this problem we are given access to a stream of documents. Each document is associated with zero or more topics from a predefined set of topics.

**Data Set Used:**

http://www.daviddlewis.com/resources/testcollections/reuters21578/
**Reuters-21578**
Source Of data set: David D. Lewis
AT&T Labs - Research

**Work distribution among the team members:**

**1) Data Preprocessing**
Done by all members
**2) Neural Network**
Sarvat Ali
Prabha Pandey
**3) KNN**
Sarvat Ali
Prabha Pandey
**4) Naive Bayes**
C Sai Sukrutha
Shruti Chandra
**5) Logistic Regression**
C Sai Sukrutha
**6) Decision Tree**
Shruti Chandra
**7) Research Work and ppt**
Done by all members

- **Data Preprocessing:**
  Two methods are used for data preprocessing for different models:
  - **TF-IDF**:-"Term Frequency – Inverse Document Frequency"

    Term Frequency: This summarizes how often a given word appears within a document.

    Inverse Document Frequency: This downscales words that appear a lot across documents.

    TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

    The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

  - **Word Counts with CountVectorizer**:-
    It tokenizes a collection of text documents and builds a vocabulary of known words, also encodes new documents using that vocabulary.
    An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

- **Techniques for Multilabel Classification:**

  **Problem transformation methods:**
  - Transform the multi-label problem into a set of binary classification problems, which can then be handled using single-class classifiers.
  - Adapt Data to the Algorithm.

  **Algorithm Adaptation Methods:**
  - Adapt a single-label algorithm to produce multi-label outputs.
  - Adapt Algorithm to the Data.

- **Problem transformation methods:**

  - **Binary Relevance:**
    An ensemble of single-label binary classifiers is trained, one for each class. Each classifier predicts either the membership or

the non-membership of one class. The union of all classes that were predicted is taken as the multi-label output. This approach is popular because it is easy to implement, however it also ignores the possible correlations between class labels.

- **Classifier Chain:**
  A chain of binary classifiers C0, C1, . . . , Cn is constructed, where a classifier Ci uses the predictions of all the classifier Cj , where j < i. This way the method, also called classifier chains (CC), can take into account label correlations.
  The total number of classifiers needed for this approach is equal to the number of classes, but the training of the classifiers is more involved.
  Following is an illustrated example with a classification problem of three categories {C1, C2, C3} chained in that order.

- **Naive Bayes:**
  - Multinomial Naive Bayes classifier uses a multinomial distribution for each of the features.
  - In the multinomial model, a document is an ordered sequence of word events, drawn from the same vocabulary V.
  - We assume that the lengths of documents are independent of class.
  - Naive Bayes assumption: that the probability of each word event in a document is independent of the word's context and position in the document
  - Thus, each document $d_i$ is drawn from a multinomial distribution of words with as many independent trials as the length of $d_i$. This yields the familiar "bag of words" representation for documents.

- **Logistic Regression:**

  For Multiclass classification, default is One-vs-Rest scheme and cross-entropy loss.

  To acheive Multi-label we used Classifier Chain.

  Advantages :

  - Provides probabilities for outcomes.

  - It is more robust: the independent variables don't have to be normally distributed

Disadvantages:

- Linear model

- **Algorithm Adaptation Methods:**

  Take any classifier, make it multi-label capable. To learn from multi-label data and make multi-label predictions.

- **KNN:**

  - Steps:
    - check cosine similarity of doc with each category.
    - labels are ranked in order of their similarity with documents.
  - Challenges:
    - Entire data is storage is needed, hence space complexity is high
    - Accuracy calculation- if order of labels is switched so should we count as true positive or not.
  - Pros: no assumptions involved as it is non-parametric.
  - Cons: slow algorithm as document size increases.
  - Improvements: parallel calculations for cosine similarity of each document and all categories to increase run time.

- **Decision Tree:**
  - Multilabel decision tree is an extension of the popular C4.5 decision tree algorithm; with multi-label entropy
  - To support multi-output problems, following changes are done:
  - Store n output values in leaves, instead of 1
  - Advantages:
    - Simple to understand and to interpret. Trees can be visualised.
    - The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
  - Disadvantages:
    - Problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
    - Decision tree learners create biased trees if some classes dominate.
  - Improvements:
    - To avoid creating over - complex tree
      - pruning should be done
      - setting the minimum number of samples required at a leaf node
      - ensemble learning mitigates the problem of unstable nature of decision tree.

- **Neural Network:**

  - We are given n samples
    $X = \{x_1, ...., x_n\}$
    and 5 labels for each category
    $y = \{y_1, ..., y_5\}$

  - We use a simple neural network as an example to model the probability $P(c_j|x_i)$ of a class $c_i$ given sample $x_i$.

    We then estimate out prediction as a vector of 5 classes, representing the probability of that sample to belong to each of the class.

  - Then first ranking is given to the class with highest probability, second ranking to the class with second highest probability and so on, hence the ranking is done of each given data sample.

  - We used sigmoid and binary_crossentropy instead of softmax and categorical_crossentropy beacuse:
    - Let the output layer give, [0.2689414213699951, 0.9933071490757153, 0.3775406687981454, 0.9933071490757153, 0.3775406687981454]
    - By using softmax, we would clearly pick class 2 and 4. But we have to know how many labels we want for a sample or have to pick a threshold. This is clearly not what we want.
    - With the sigmoid activation function at the output layer the neural network models the probability of a class c_j as Bernoulli distribution. Now the probabilities of each class is independent from the other class probabilities. So we can use the threshold 0.5 as usual.
    - We use the binary_crossentropy loss and not the usual in multi-class classification used categorical_crossentropy loss, as we want to penalize each output node independently. So we pick a binary loss and model the output of the network as a independent Bernoulli distributions per label.

- **Future Scope:**

  - Problem with our data preprocessing:
    Counts and frequencies can be very useful, but one limitation of these methods is that the vocabulary can become very large.

This, in turn, will require large vectors for encoding documents and impose large requirements on memory and slow down algorithms.

- Solution for this problem:

1) Hashing with HashingVectorizer:-

Use a one way hash of words to convert Counts and frequencies to integers.

No vocabulary is required, as hash of words is used.

A downside is that the hash is a one-way function so there is no way to convert the encoding back to a word.

2) word2Vec:

In general, when you like to build some model using words, simply labeling/one-hot encoding them is a plausible way to go. However, when using such encoding, the words lose their meaning.

Solution: A numeric representation for each word, that will encapsulate different relations between words, like synonyms, antonyms, or analogies.

How is it done: There are two methods to do this:

1) Continuous bag of words:

Continuous bag of words creates a sliding window around current word, to predict it from “context”—the surrounding words.
Each word is represented as a feature vector. After training, these vectors become the word vectors.
Vectors which represent similar words are close by different distance metrics, and additionally encapsulate numeric relations.

2) Skip gram:

Opposite of CBOW, instead of prediciting one word each time, we use 1 word to predict all surrounding words (“context”). But it is much slower than CBOW. Though more accurate with infrequent words.

- **Bibliography:**


  - http://www.jmlr.org/papers/volume3/crammer03b/crammer03b.pdf
  - https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
  - https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/
  - https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
  - https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff
  - https://en.m.wikipedia.org/wiki/Cosine_similarity
  - https://scikit-learn.org/stable/modules/tree.html#tree
  - https://scikit-learn.org/stable/modules/multiclass.html