**Report of the Practical Assignment in Information Retrieval Course**

**Edited by:**

Tayma Ibrahem

Oron Kam

***Task 1 -*** Language modelling for the directory "Train and Test".

For this Task we have used the existing python libraries like ‘nltk’ to build a unigram model where each word's probability is estimated using the Simple Good-Turing method based on its frequency in the training texts.

**The Simple Good-Turing probability distribution model**, implemented through NLTK, offers a sophisticated approach to estimating probabilities in a unigram language model. Leveraging the frequency distribution of words within a corpus, this model assigns probabilities to each word occurrence using the principles of the Good-Turing estimation method. By accounting for the unseen or infrequent words through statistical smoothing, it addresses the limitations of traditional maximum likelihood estimation, thus enhancing the model's accuracy and robustness in predicting word probabilities. This technique proves particularly beneficial for language modelling, where understanding the likelihood of individual word occurrences is paramount. so for conclusion the model that we provided above is unigram LM model with smoothing.

**break down each step of the LM model building and explain what it does:**

**Step 1**: Read the text files from the directory, which is performed just once to load the data for the next operations.

**Step 2:** Tokenize the text data.

* Tokenization is the process of splitting text into individual words or tokens.
* It uses the word\_tokenize() function from the NLTK library to tokenize each text in the texts list(loaded documents).
* After tokenization, it removes duplicate tokens from each text.
* **In this step we have evaluate the model 4 times one for each case as the following:**

1. **LM with general approach without Performing a set of linguistic operations.**
2. **The same model but with performing Case folding.**

performing case folding (converting text to lowercase) to ensure uniformity and then tokenizes the lowercased text. By removing duplicates before tokenization, it helps streamline the vocabulary and ensures that each token represents a unique word.

1. **The same model but with performing Stemming.**

performs stemming to reduce words to their base form and then tokenizes the stemmed text. using the Porter stemming algorithm, which is a popular stemming algorithm. By applying stemming before tokenization, the code ensures that each token represents a unique stemmed word. This preprocessing step helps normalize the vocabulary and reduce redundancy, making the text data more suitable.

1. **The same model but with removing Stop Words.**

The code begins by creating a set of stop words using the NLTK library's stopwords.words('english') function, which provides a list of common English stop words.

It then iterates over each text in the texts list and removes any words that are found in the set of stop words.

removes stop words from the text data and then tokenizes the filtered text. By removing stop words before tokenization, the code ensures that the resulting tokens only include meaningful words, excluding common stop words that often don't contribute much to the semantics of the text. This preprocessing step helps streamline the vocabulary and improve the quality of the text data for tasks.

**\*Each step of the following performed for every scenario in the tokenization phase:**

**Step 3:** Split the dataset into training, testing, and validation sets.

* This step involves splitting the tokenized dataset into training, testing, and validation sets.
* It uses the train\_test\_split() function from scikit-learn to split the data.
* 80% of the data is allocated for training (train\_texts), and the remaining 20% is split equally between testing and validation (test\_texts and val\_texts).

**Step 4:** Train a unigram language model.

* In this step, a unigram language model is trained using the training data.
* The unigram model is created using the Simple Good-Turing probability distribution provided by NLTK.
* It calculates the frequency distribution of words in the training data and uses it to estimate probabilities for each word.

**Step 5:** Test the model.

* The trained unigram model is tested using the testing data to calculate perplexity.
* Perplexity is a measure of how well the model predicts the test data. Lower perplexity indicates better performance.
* The perplexity() computes the perplexity of the test data given the language model.

**about perplexity** - the measurement that we evaluate the performance of the model based of it:

Perplexity is a metric used to evaluate the performance of language models, such as n-gram models. It measures how well a probability distribution or language model predicts a sample.

Perplexity is defined as:

A white background with black text

Description automatically generated

lower perplexity values indicate better performance. A perplexity value closer to 1 indicates that the model is very certain about its predictions and is close to the true underlying distribution of the data. However, it's essential to consider the context and complexity of the dataset when interpreting perplexity values. In practice, perplexity values can range from less than 10 to several hundred or even thousands.

**For example:**

Perplexity values below 100 are often considered good for language modelling tasks.

Perplexity values above 100 indicate that the model may struggle to accurately predict the next word in the sequence.

Extremely high perplexity values (e.g., in the thousands) suggest that the model is performing poorly and may not capture the underlying patterns in the data effectively.

**Step 6:** Perform validation and print the result.

* Similar to the testing step, the validation data is used to evaluate the performance of the model.
* The perplexity of the validation data is computed and printed.
* The result gives an indication of how well the model generalizes to unseen data. Lower perplexity suggests better generalization.
* **Result Interpretation of all linguistic operations as described in Step 2 above:**
  1. **LM with general approach without Performing a set of linguistic operations.**
* Test Perplexity: 834.79
* Validation Perplexity: 841.84
  + This represents the performance of the unigram model without any linguistic preprocessing. The high perplexity score indicates that the model struggles to accurately predict the next word in the test data, due to the presence of noise and variability in the text, and likely because of a lot of stop words that increase the vocabulary size and the complexity of the model without adding much value, The same is due to not implementing case folding and without stemming or another optimization processes which lead to a smaller and cleaner vocabulary that focuses on the core content of the collection.
    1. **The same model but with performing Case folding.**
  + Test Perplexity: 94.130
  + Validation Perplexity: 91.547
  + Applying case folding helps normalize the text by converting all characters to lowercase. This reduces the variability caused by different cases of the same word, leading to improved performance compared to the baseline. The significant decrease in perplexity suggests that case folding contributes to better word prediction.
    1. **The same model but with performing Stemming.**
  + Test Perplexity: 24.220
  + Validation Perplexity: 24.295
  + Stemming reduces words to their root or base form, thereby collapsing variations of the same word into a single representation. This operation further reduces the vocabulary size and helps the model generalize better across morphological variations. The substantial decrease in perplexity indicates that stemming has a significant positive impact on the model's performance, more than case folding.
    1. **The same model but with removing Stop Words.**
  + Test Perplexity: 19.287
  + Validation Perplexity: 19.617
  + Stop words, which are common and non-discriminative words like "the," "and," "is," etc., often add noise to language models without contributing much to their predictive power. Removing these words reduces the noise in the data and allows the model to focus on more informative words. The lowest perplexity score among all iterations suggests that stop word removal has the most profound effect on improving the model's predictive performance. In summary, each linguistic operation contributes to reducing the complexity and noise in the text data, leading to improved performance of the unigram model in predicting word sequences. While case folding, stemming, and removing stop words all play important roles, stop word removal appears to have the most significant impact on reducing perplexity and enhancing the model's predictive capabilities.

**For Conclusion:**

Each linguistic operation contributes to reducing complexity and noise in the text data, leading to improved model performance.

While case folding, stemming, and removing stop words all play important roles, stop word removal appears to have the most significant impact on reducing perplexity and enhancing the model's predictive capabilities.

Overall, the combination of all linguistic operations (case folding, stemming, and stop words removal) would likely result in the best performance by reducing noise, normalizing text, and improving the model's ability to generalize across different word forms and variations.

Several outside factors can influence the performance of a language model, beyond the linguistic operations applied to the data. Here are some key external factors:

**Quality and Quantity of Training Data:** The quality and size of the training dataset significantly impact the model's performance. Larger and more diverse datasets generally lead to better generalization and performance.

**Data Preprocessing Techniques**: Other data preprocessing techniques, such as handling punctuation, dealing with special characters, or encoding text, can affect the model's performance. Proper data cleaning and normalization are crucial for effective training.

**Model Architecture and Hyperparameters:** The choice of model architecture (e.g., unigram, bigram, neural network-based models) and hyperparameters (e.g., learning rate, batch size) can greatly influence the model's ability to learn patterns from the data.

**Domain Specificity:** Language models trained on specific domains may not generalize well to other domains. Consideration of the domain of the text data and tailoring the model accordingly is important for optimal performance.

Language and Text Complexity: The complexity of the language and text data being modeled can affect the model's performance. Complex grammatical structures, rare words, or domain-specific jargon may pose challenges for the model.

***Task 2 -*** Text classification.

For this Task we have used the existing python libraries like ‘nltk’ to build a classification tasks to try to predict the documents classes.

We start by loading all the documents with their corresponding labels and we split them into train and test in a 90/10 precentages for train and test accordingly.

**Classification models chosen and results:**

We choose 4 classification models for this task including logistic regression, naïve bayes, svm and random forest.

**Evaluation:**

We choose to evaluate the model based on those parameters: Accuracy,Precision,recall which are relevant values in terms of documents retrieval and in information retrieval in general.

**Logistic regression:**

Despite achieving perfect accuracy, precision and recall of 1.0 logistic regression may be considered cautiously due to its performance being assessed solely on these metrics.

While its simplicity and high performance are apparent, its important to scrutinize its generalization to unseen data and its capabilities in handling more complex patterns.

**Naive Bayes:**

Similar to logistic Regression, Naïve Bayes also achieved perfect accuracy,precision and recall of 1.0, indicating consistent and reliable performance across all metrics. This makes Naïve Bayes an attractive choise for tasks where the robustness and simplicity are valued, and where the assumptions of independence among features hold reasonably well.

**SVM**:

While SVM falls behind in terms of accuracy and recall compared to Logistic Regression and Naïve Bayes, it still demonstrates a respectable performance with accuracy and recall values of 0.5455 which indicates a little bit understanding of the data (more then 50%). how ever its precision of 0.7980 suggests that its relatively better at minimizing false positive at the expense of some false negatives. Therefore SVM might be suitable option when false positives need to be minimized while maintaining a decent level of recall.

**Random Forest:**

Random forest acgived an accuracy of 0.9091 and precision of 0.9273 with recall matching its accuracy. Although not perfect, its performance is commendable, especially considering its robustness and ability to handle complex relationships in the data. Random Forest could be a preferable choice when high accuracy and reliability in classifiying positive cases are prioritized without the strict requirement of perfect performance on all metrics.

**Summery:**

In summary the choice of the best classifier depends on the specific needs of the task. Logistic regression and Naive Bayes offers simplicity and perfect performance but may require further evaluation for generalization. SVM with its focus on minimizing false positives, could be valuable in scenarios where this is crucial. Random Forest, with its robustness and strong performance, may be a suitable option for tasks requiring high accuracy and reliability.

***Task 3 -*** Text clustering.

**break down each step of this Task:**

**Step 1:** Load documents from directories

The code loads text documents from specified directories and assigns labels to each document based on the directory it was loaded from.

**Step 2:** Combine all documents and labels into a single list

The documents and labels from all directories are concatenated into single lists (all\_documents and all\_labels, respectively).

**Step 3:** Vectorize the text using TF-IDF

* The text data is vectorized using the Term Frequency-Inverse Document Frequency (TF-IDF) representation.
* This converts the text data into numerical form, where each document is represented as a vector of TF-IDF values for each word in the vocabulary.
* Stop words (common words like "the," "and," etc.) are removed during this process using the **'english'** parameter.

**Step 4:** Reduce the dimensionality of the data using PCA

Principal Component Analysis (PCA) is applied to reduce the dimensionality of the TF-IDF vectors to 2 dimensions.

This step is performed to facilitate visualization of the data in a 2D space.

**Step 5**: Cluster the documents using k-means

* The k-means clustering algorithm is applied to cluster the documents into **num\_clusters** clusters.
* The **num\_clusters** parameter is set to 4 in this case.

**Step 6:** Evaluate clustering performance using Adjusted Rand Index (ARI)

The Adjusted Rand Index (ARI) is calculated to measure the similarity between the true labels and the cluster assignments.

**The Adjusted Rand Index (ARI):**

is a measure used to evaluate the similarity between clusterings. It compares how well the clusters obtained from an algorithm match the true clusters or ground truth labels.

The ARI value ranges from -1 to 1:

A score of 1 indicates perfect agreement between theclusterings, meaning that they are identical. A score close to 0 indicates that the clusterings are random or do not agree better than random chance. A negative score indicates that the agreement is worse than random chance.

the Adjusted Rand Index provides a way to quantify the similarity between clusterings, allowing us to assess the performance of a clustering algorithm compared to a ground truth or reference clustering.

**\*After training our model we have got that Adjusted Rand Index (ARI): 0.669064552801996.**

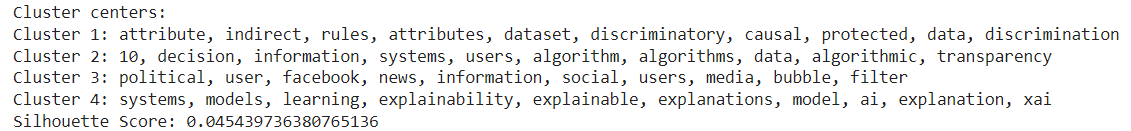
**Step 7**: Visualization of the clustering result

* The clustering results are visualized using a scatter plot.
* Each point represents a document, and the color of each point corresponds to its assigned cluster.
* The legend indicates the cluster number.
* The cluster assignments for 5 document are printed and plotted, for example.

A screen shot of a computer screen

Description automatically generated

**Step 8:** Calculate Silhouette Score

* The Silhouette Score is calculated to measure the quality of the clustering.
* The top 10 features (words) for each cluster centre are printed.
* These features represent the most important terms within each cluster.
* **Silhouette Score** : is an another evaluation method of the model.
* measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It ranges from -1 to 1, where:
* A score close to +1 indicates that the sample is far away from the neighboring clusters. A score of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters. A score close to -1 indicates that the sample is misclassified and is more similar to a neighboring cluster than its own cluster. Therefore, the higher the silhouette score, the better results we get.
* The result of this step is :

***Analyse the results of the clustering and Explain errors:***

According to what we saw in the plot above, the data are distributed in space in such a way that there are three main centres, and this explains the evaluation score that we obtained of 0.666 which is good but not enough because we choose k=4 and we wanted to cluster it into 4 clusters according what we think because of the division form we received, where as opposed naturally the distribution of data is 3. Therefore, if we activate the model again with K = 3 definitely we will get better results. another reason of the results is that the clusters are Non-Globular Clusters as we see above . where K-means performs poorly on non-globular clusters, as it tends to create spherical clusters around centroids.

and at last also because of the content of the documents, it's obviously that there is common content in the documents in cluster 2,3,4, which lead to bias.as well as we can see this in the top 10 terms of each cluster there is common terms in these clusters.