**W10 – NLP -Hate Speech Detector**

| **Criteria** |
| --- |
| Execute Programming Code Assignment  Complete Tasks 1-6 in the Hate Speech Detector Notebook |
| **Algorithm Understanding**  **How does the Naive Bayes Classifier work? What is Posterior Probability?**  Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of conditional independence between every pair of features given the value of the class variable. Bayes’ theorem states the following relationship: **posterior probability** = (conditional probability \* prior probability) / evidence.  Given a class variable y and a dependent feature vector x1 through xn,  this can be written as P(y | x1, …., xn) = P(x1, …, xn | y) \* P(y) / P(x1, … , xn). The posterior probability, in the context of a classification problem, can be interpreted as: “What is the probability that a particular object belongs to class y given its observed feature values xi?”  Another assumption that Bayes classifiers make is that the samples are “independent and identically distributed” (iid) like random variables that are independent from one another and are drawn from a similar probability distribution. Independence means that the probability of one observation does not affect the probability of another observation (e.g., time series and network graphs are not independent). One popular example of idd variables is the classic coin tossing: The first coin flip does not affect the outcome of a second coin flip and so forth. Given a fair coin, the probability of the coin landing on “heads” is always 0.5 no matter of how often the coin if flipped.  Under the assumption of conditional independence of features, the class-conditional probabilities or (likelihoods) of the samples can be directly estimated from the training data instead of evaluating all possibilities of x. Using this naïve assumption: for every i then the conditional probability is = P(xi | y) and the total conditional probability is just the product of P(xi | y) for i varying from 1 to n. As we are looking for a classification rule, and the evidence term (P(x1, … , xn).) is constant, we can eliminate it from the classification formula. P(y) can be calculated as the relative frequency of class y in the training set. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi∣y).  In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality. On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously. |
| **Interview Readiness**  **What is the difference between stemming and lemmatization in NLP?**  Stemming is the process of reducing words (inflected or sometimes derived) to their word stem, base, or root form. For example, if there are two words in the corpus `walks` and `walking`, then stemming will stem the suffix to make them `walk`. But say in another example, we have two words `console` and `consoling`, the stemmer will remove the suffix and make them `consol` which is not a proper English word.  Lemmatization is similar to stemming in reducing inflected words to their word stem but differs in the way that it makes sure the root word (also called as lemma) belongs to the language. As a result, this process is generally slower than the stemming process. So, depending on the speed requirement of the project at hand, we can choose to use either stemming or lemmatization. The lemmatization process also depends on the POS tag to come up with the correct lemma. As an example, if we are getting the root form `run` we also need to provide the POS tag of the word (if it is a verb or a noun, or both) along with the word to the lemmatizer (ea. in nltk). Depending on the POS, the lemmatizer may return different results. |
| **Interview Readiness**  **What is Word2Vec and how does it work?**  The idea behind w2v is that if 2 words are similar in meaning, they should be ‘closer’ to each other compared to 2 words that are not. While one hot encoding provides very sparse vectors (a los of zeros), and even sparser matrixes, w2v provides a dense representation of words (or n grams, n being the number of words in the token) by using word embeddings.  W2v associates each word to a ‘position’ in the embedding space by learning the representation of words as vectors in the multi-dimensional embedding space. Then w2v calculates similarities between words by estimating the cosine of the ‘angle’ formed by the vector representing those two words in the multi-space of word embeddings. The idea behind this is that the closer in meaning they are, the closer to zero that angle should be (or that the projection of one word-vector on the other will be closer to its full ‘size’). By representing word as vectors, w2v can also perform vector addition and subtraction between related word-vectors and apply that space difference to other words to predict a similarly related word.  W2v is a statistical method, that does a self-supervised training to develop the word embeddings for a text corpus. W2v is usually trained using a bag-of-words model (CBOW with or without tfidf) or a skip gram model. |
| **Interview Readiness**  **When to use GRU over LSTM?**  The key difference between a GRU (gated recurrent unit) and an LSTM (long short term memory), two recurrent neural networks, is that a GRU has two gates (reset and updategates) whereas an LSTM has three gates (input, output and forget gates).  GRUs are generally used when you do have long sequence training samples and you want a quick and decent accuracy and maybe in cases where infrastructure is an issue. LSTMs are preferred when sequence lengths are more and some good context is there. LSTMs when trained with more data give you better results than GRUs.  GRUs train faster and perform better than LSTMs on less training data if you are doing language modeling. GRUs are simpler and thus easier to modify, for example adding new gates in case of additional input to the network. It's just less code in general.  LSTMs should, in theory, remember longer sequences than GRUs and outperform them in tasks requiring modeling long-distance relations. |