Question 1

HTSPC- Hate Speech Classification

Problem statement: We are given a piece of text, which we need to classify into hate speech or not hate speech. It is a binary classification problem with labels **“HOF”(0)** denoting hate speech and **“NOT”(1)** denoting non-hateful sentences.

In this question, we have to classify the comment whether it is a hate comment or not.

To do this we first need to preprocess the data, remove irrelevant things like URLs, etc which doesn’t tell whether the speech is hate or not.

Preprocessing

1. Lower case data.
2. Remove URLs
3. **Remove punctuations**
   Removes this set of symbols [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~]
4. Remove numbers
5. **Remove stopwords**
   “Stop words” are the most common words in a language like “the”, “a”, “on”, “is”, “all”. These words do not carry important meaning and are usually removed from texts. It is possible to remove stop words using Natural Language Toolkit (NLTK), a suite of libraries and programs for symbolic and statistical natural language processing.
6. **Stemming**
   Stemming is a process of reducing words to their word stem, base, or root form (for example, books — book, looked — look).
7. **Lemmatization**
   The aim of lemmatization, like stemming, is to reduce inflectional forms to a common base form. As opposed to stemming, lemmatization does not simply chop off inflections. Instead, it uses lexical knowledge bases to get the correct base forms of words.

```
def remove_stopwords(sentence):
 token_words=word_tokenize(sentence)
 stem_sentence=[]
 for word in token_words:
     if not word in stopwords.words():
       stem_sentence.append(word)
 return " ".join(stem_sentence)
```

```
def stemSentence(sentence):
   token_words=word_tokenize(sentence)
   # token_words
   stem_sentence=[]
   for word in token_words:
     stem_sentence.append(porter.stem(word))
   return " ".join(stem_sentence)

def lemmantize(sentence):
 token_words=word_tokenize(sentence)
 #print(token_words)
 stem_sentence=[]
 for word in token_words:
   stem_sentence.append(lemmatizer.lemmatize(word))
 return " ".join(stem_sentence)


def camel_case_split(string):
   return re.findall('[A-Z][^A-Z]*', string)

def pre_process(X):
 list_X=[]
 for i in range(X.shape[0]):
   #print(i)
   comment=X[i]
   comment=comment.lower()
   #print(comment)
   comment = re.sub(r"http\S+", "", comment)
   #print(comment)
   comment=remove_punctuation(comment)
   comment=''.join([i for i in comment if not i.isdigit()])
   #print(comment)
   comment=remove_stopwords(comment)
   #print(comment)
   #comment=lemmantize(comment)
   #print(comment)
   comment=stemSentence(comment)
   #print(comment)
   list_X.append(comment)
```

I have not removed Emojies because It affects the predictions.

Also, I have converted sentences like **“ThatIsABadThing” into “that is a bad thing” by** camel_case_split().

To convert sentences to vector, I used Tf-Idf vectorizer.

Tf-Idf

Tf-Idf stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining.
This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.
This is the formula for Tf-Idf:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

where,

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \qquad idf(w) = log(\frac{N}{df_t})$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer()
```

Models used :

1. **Logistic regression**

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0).fit(X_train_tfidf, y)
predicted=clf.predict(X_test_tfidf)
```

2. **Support Vector machines (SVM)**

```
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train_tfidf, y)
y_pred = classifier.predict(X_test_tfidf)
```

Observations:

1. *From Logistic regression, I got 83.45 % validation accuracy.*
2. *From SVM with ‘rbf’ kernel and C=1.0, I got 95.55 % validation accuracy.*
3. *Preprocessing highly affecting the accuracy. I have considered many things like emojis and camel case words during the preprocessing.*

Conclusion:

Hence, For test data, I have used SVM to train the model and I got 85.3% test accuracy.

.