TOXIC COMMENT CLASSIFICATION

Problem Statement: To identify and classify toxic online comments. To build a multi-headed model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate.

Introduction:

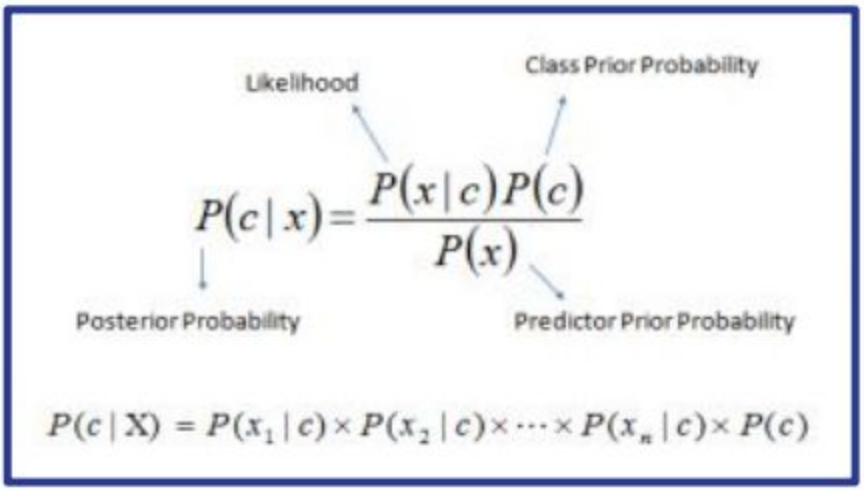
Algorithm used for Data Preprocessing:

1. **Tokenizer:** Tokenization is the process of taking text (such as a sentence) and breaking it into individual terms (usually words). A simple Tokenizer class provides this functionality.
2. **HashingTF:** HashingTF is a Transformer which takes sets of terms and converts those sets into fixed-length feature vectors. HashingTF utilizes the hashing trick. A raw feature is mapped into an index (term) by applying a hash function.Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e. the number of buckets of the hash table.
3. **Term frequency-inverse document frequency (TF-IDF) algorithm:** TF-IDF is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus.In the algorithm, we start with a set of sentences. We split each sentence into words using Tokenizer. For each sentence (bag of words), we use HashingTF to hash the sentence into a feature vector. We use IDF to rescale the feature vectors; this generally improves performance when using text as features. Our feature vectors could then be passed to a learning algorithm.

Algorithm used for Data Classification:

1. **Naive Bayes:** Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction.
2. **Multinomial:** It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in

the document", you can think of it as "number of times outcome number x_i is observed over the n trials".



Approach:

1. Java and Hadoop Setup in Google Collaboratory
2. Pyspark 2.3 installation
3. Reading the train.csv and test.csv using pandas
4. Creating dataframe using hive context
5. Performing data pre processing and applying classification algorithm Naive Bayes
6. Using Naive Bayes predicting the class of the comment text in test data set and generating output file.

Code:

```
import pandas as pd
from pyspark import SparkConf, SparkContext, HiveContext
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.ml.feature import Tokenizer, HashingTF, IDF
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import DecisionTreeClassifier
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
file_id = '1bDRfC70G-6bOgzkMw8R6UfrQN8sEUlX8'
```

```
downloaded = drive.CreateFile({'id': file_id})
downloaded.GetContentFile("train.csv")
conf = SparkConf().setAppName("Cloudera Project")
sc = SparkContext(conf=conf)
hc = HiveContext(sc)
dftrain=pd.read_csv("train.csv")
train=hc.createDataFrame(dftrain)
dftest=pd.read_csv("test.csv")
test=hc.createDataFrame(dftest)
classes = [i for i in train.columns if i not in ["id", "comment_text"]]
tokenizer = Tokenizer(inputCol="comment_text", outputCol="words")
wordsData = tokenizer.transform(train)
hashingTF = HashingTF(inputCol="words", outputCol="hashdata")
tf = hashingTF.transform(wordsData)
idf = IDF(inputCol="hashdata", outputCol="features")
idfModel = idf.fit(tf)
tfidf = idfModel.transform(tf)
test_tokens = tokenizer.transform(test)
test_tf = hashingTF.transform(test_tokens)
test_tfidf = idfModel.transform(test_tf)
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(smoothing=1.0, modelType="multinomial",labelCol='toxic',featuresCol="features")
model = nb.fit(tfidf)
predictions = model.transform(test_tfidf)
extract_prob = F.udf(lambda x: float(x[1]), T.FloatType())
test_probs = []
for col in classes:
    print(col)
    nb = NaiveBayes(smoothing=1.0, modelType="multinomial",labelCol=col,featuresCol="features")
    nbModel = nb.fit(tfidf)
    res = nbModel.transform(test_tfidf)
    test_res = test_res.join(res.select('id', 'probability'), on="id")
    test_res = test_res.withColumn(col, extract_prob('probability')).drop("probability")
    test_res.show()
test_res_pan = test_res.toPandas()
test_res_pan.head(10)
```

Output:

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	000968ce11f5ee34	8.949112e-12	2.923570e-37	1.509294e-10	2.178481e-33	1.121275e-13	9.224082e-33
1	000bafe2080bba82	1.381710e-27	0.000000e+00	6.047047e-31	0.000000e+00	2.048829e-30	1.434845e-34
2	0009734200a85047	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
3	0003806b11932181	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
4	000bf0a9894b2807	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
5	00091c35fa9d0465	1.121513e-10	0.000000e+00	2.217137e-09	0.000000e+00	4.744609e-07	8.612464e-09
6	00025358d4737918	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
7	000834769115370c	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
8	0002eadc3b301559	1.039787e-10	8.407791e-45	2.898816e-13	5.430032e-42	1.029478e-14	1.249761e-27
9	00001cee341fdb12	1.000000e+00	6.789010e-24	1.000000e+00	2.873344e-29	1.000000e+00	1.000000e+00

Conclusion: Thus, we have identified and classified online toxic comments using Naive Bayes Classifier.