In [1]:
```
import pandas as pd
import numpy as np
from collections import defaultdict
import re
```

Cleaning function:

In [2]:
```
def preprocess_string(str_arg):

    cleaned_str=re.sub('[^a-z\s]+',' ',str_arg,flags=re.IGNORECASE)
    cleaned_str=re.sub('(\s+)',' ',cleaned_str)
    cleaned_str=cleaned_str.lower()

    return cleaned_str
```

Naive Bayes from scratch:

In [3]:
```
class NaiveBayes:

    def __init__(self,unique_classes):
        self.classes=unique_classes


    def addToBow(self,example,dict_index):

        if isinstance(example,np.ndarray): example=example[0]

        for token_word in example.split():
            self.bow_dicts[dict_index][token_word]+=1


    def fit(self,dataset,labels):
        self.examples=dataset
        self.labels=labels
        self.bow_dicts=np.array([defaultdict(lambda:0) for index in ran
ge(self.classes.shape[0])])

        for cat_index,cat in enumerate(self.classes):
            all_cat_examples=self.examples[self.labels==cat]
            cleaned_examples=[preprocess_string(cat_example) for cat_ex
ample in all_cat_examples]
            cleaned_examples=pd.DataFrame(data=cleaned_examples)
            np.apply_along_axis(self.addToBow,1,cleaned_examples,cat_in
dex)

        prob_classes=np.empty(self.classes.shape[0])
        all_words=[]
        cat_word_counts=np.empty(self.classes.shape[0])


        for cat_index,cat in enumerate(self.classes):
            prob_classes[cat_index]=np.sum(self.labels==cat)/float(self
.labels.shape[0])
            count=list(self.bow_dicts[cat_index].values())
            cat_word_counts[cat_index]=np.sum(np.array(list(self.bow_di
cts[cat_index].values())))+1
            all_words+=self.bow_dicts[cat_index].keys()

        self.vocab=np.unique(np.array(all_words))
        self.vocab_length=self.vocab.shape[0]

        denoms=np.array([cat_word_counts[cat_index]+self.vocab_length+1
 for cat_index,cat in enumerate(self.classes)])

        self.cats_info=[(self.bow_dicts[cat_index],prob_classes[cat_ind
ex],
                         denoms[cat_index]) for cat_index,cat in enumer
ate(self.classes)]
        self.cats_info=np.array(self.cats_info)
```

```
    def getExampleProb(self,test_example):

        likelihood_prob=np.zeros(self.classes.shape[0])
        for cat_index,cat in enumerate(self.classes):
            for test_token in test_example.split():

                test_token_counts=self.cats_info[cat_index][0].get(test
_token,0)+1
                test_token_prob=test_token_counts/float(self.cats_info[
cat_index][2])
                likelihood_prob[cat_index]+=np.log(test_token_prob)
        post_prob=np.empty(self.classes.shape[0])


        for cat_index,cat in enumerate(self.classes):
            post_prob[cat_index]=likelihood_prob[cat_index]+np.log(self
.cats_info[cat_index][1])
        return post_prob


    def predict(self,test_set):

        predictions=[]
        for example in test_set:
            cleaned_example=preprocess_string(example)

            post_prob=self.getExampleProb(cleaned_example)
            predictions.append(self.classes[np.argmax(post_prob)])
        return np.array(predictions)
```

Training data-set:

In [4]:
```
training_set=pd.read_csv ("train.csv")
training_set.info()
training_set.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   id         60000 non-null  int64 
 1   comment    60000 non-null  object
 2   subreddit  60000 non-null  object
dtypes: int64(1), object(2)
memory usage: 937.6+ KB
```

Out[4]:
```
(60000, 3)
```

In [5]:
```
training_set.head()
```

Out[5]:

	id	comment	subreddit
0	0	I think prestige points should not expire ever...	leagueoflegends
1	1	Whats going to happen with them if they will b...	europe
2	2	Anecdotal evidence is anecdotal. Clearly by “e...	gameofthrones
3	3	Look dude, with all due respect, your music is...	Music
4	4	Hope he gets the doomhammer back!	wow

Prediction on train data-set:

In [6]:
```
y_train=training_set['subreddit'].values
x_train=training_set['comment'].values


from sklearn.model_selection import train_test_split
train_data,test_data,train_labels,test_labels=train_test_split(x_train,
y_train,
                                                                          shuffle=
True,
                                                                  test_siz
e=0.2
                                                                  ,random_
state=1
                                                                  ,stratif
y=y_train)
classes=np.unique(train_labels)

nb=NaiveBayes(classes)
nb.fit(train_data,train_labels)

y_pred_train=nb.predict(test_data)
test_acc=np.sum(y_pred_train==test_labels)/float(test_labels.shape[0])

print ("Test Set Accuracy: ",test_acc)
```

```
Test Set Accuracy:  0.5183333333333333
```

Testing data-set:

In [7]:

```
testing_set=pd.read_csv ("test.csv")
testing_set.info()
testing_set.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   id       20000 non-null  int64 
 1   comment  20000 non-null  object
dtypes: int64(1), object(1)
memory usage: 234.4+ KB
```

Out[7]:

```
(20000, 2)
```

In [8]:

```
testing_set.head()
```

Out[8]:

	id	comment
0	0	Holy shit a shot counter.
1	1	It doesn't matter that it isn't hard to rememb...
2	2	I find it funny that this is downvoted
3	3	They are really getting ridicoulous with all t...
4	4	He's Eden's best friend

Prediction on test data-set:

In [9]:
```
X_test=testing_set.comment.values

y_pred_test=nb.predict(X_test)
```

Submission:

In [10]:
```
submission = zip(list(range(len(y_pred_test))), y_pred_test)
test_df = pd.DataFrame(submission, columns=['Id','Category'])
test_df.to_csv('submission.csv', index = False, header=True)
```

Naive Bayes cross validation by using KFold:

In [11]:
```
from sklearn.model_selection import KFold

X = x_train
y = y_train
kf = KFold(n_splits=5, random_state=42, shuffle=True)

print(kf)
```

```
KFold(n_splits=5, random_state=42, shuffle=True)
```

In [12]:
```
main_cross_val_accuracy = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    nb.fit(X_train,y_train)
    y_pred_cross_train=nb.predict(X_test)
    test_cross_acc=np.sum(y_pred_cross_train==y_test)/float(y_test.shap
e[0])
    main_cross_val_accuracy.append(test_cross_acc)
```

In [13]:
```
main_cross_val_accuracy
```

Out[13]:
```
[0.4533333333333333,
 0.44683333333333336,
 0.45108333333333334,
 0.45466666666666666,
 0.4489166666666667]
```

#

Preparing the data for selective classifiers:

In [14]:
```
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

def clean_text(training_set):

    all_comments = list()
    lines = training_set["comment"].values.tolist()
    for text in lines:
        text = text.lower()
        pattern = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!
*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
        text = pattern.sub("", text)


        text = re.sub(r"[,.\"!@#$%^&*(){}?/;`~:<>+=-]", "", text)

        tokens = word_tokenize(text)

        table = str.maketrans('', '', string.punctuation)

        stripped = [w.translate(table) for w in tokens]
        words = [word for word in stripped if word.isalpha()]

        stop_words = set(stopwords.words("english"))
        stop_words.discard("not")

        words = [w for w in words if not w in stop_words]
        words = ' '.join(words)

        all_comments.append(words)
    return all_comments

all_comments = clean_text(training_set)
all_comments[0:2]
```

Out[14]:

```
['think prestige points not expire ever skins buy available set duratio
n exemple year release another skin vault old one making also limiteded
ition skin also please love god nt rerelease skins need grind prestige
shop would suck everyone grinded',
 'whats going happen refused asilum appeal']
```

Most frequent used words

In [15]:

```
from nltk.probability import FreqDist
from nltk.stem import WordNetLemmatizer
```

```
c = all_comments
filtered_sentence = []
freq_count_limit = FreqDist()
lemmatizer=WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

for i in c:
    comment_tokens = word_tokenize(i)

    for words in comment_tokens:
        if words not in stop_words:
            filtered_sentence.append(words)

            limit_words = lemmatizer.lemmatize(words)
#     for word in root_words:
            freq_count_limit[limit_words.lower()]+=1
freq_count_limit
```

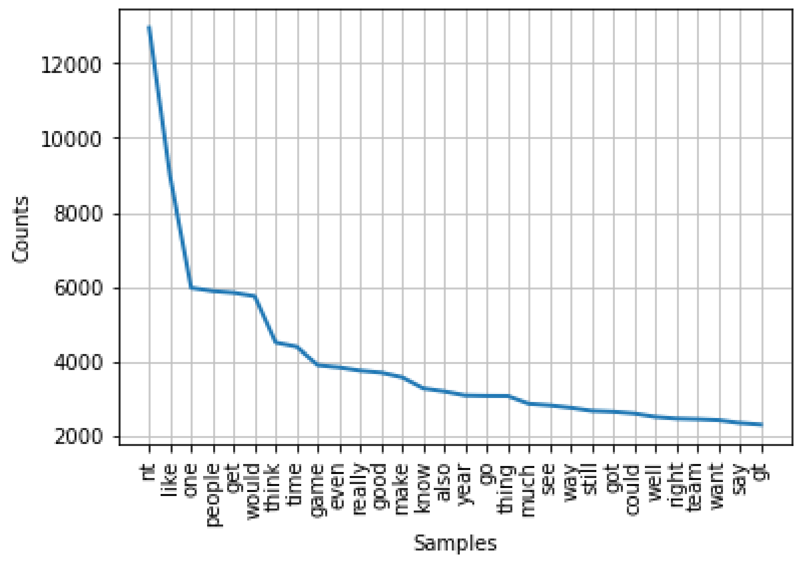
Out[15]:

```
FreqDist({'nt': 12955, 'like': 8953, 'one': 5971, 'people': 5884, 'ge
t': 5839, 'would': 5745, 'think': 4498, 'time': 4385, 'game': 3900, 'ev
en': 3839, ...})
```

In [16]:

```
import matplotlib.pyplot as plt

freq_count_limit.plot(30,cumulative=False)
plt.show()
```



Vectorizing and transforming the text:

In [17]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vect = TfidfVectorizer(ngram_range=(1,1), max_features=30000, strip_acc
ents='ascii')
vect.fit(all_comments)
vocabulaire = vect.get_feature_names()
```

In [18]:
```
bag_of_words = vect.transform(all_comments)
bag_of_words.shape
```

Out[18]:
```
(60000, 30000)
```

Random Forest:

In [19]:
```
from sklearn.ensemble import RandomForestClassifier
```

```
clf_rf = RandomForestClassifier(max_depth=15, random_state=42)
clf_rf.fit(bag_of_words, training_set['subreddit'])
```

```
Out[19]: RandomForestClassifier(max_depth=15, random_state=42)
```

```
In [20]: clf_rf.score(bag_of_words, training_set['subreddit'])
```

```
Out[20]: 0.34563333333333335
```

Random Forest - Cross Validation:

```
In [21]: from sklearn.model_selection import cross_val_score

scores_rf = cross_val_score(clf_rf, bag_of_words, training_set['subredd
it'], cv=5)
scores_rf
```

```
Out[21]: array([0.30675   , 0.30808333, 0.31591667, 0.31841667, 0.31941667])
```

Logistic Regression:

```
In [22]: from sklearn.linear_model import LogisticRegression

clf_lr = LogisticRegression(max_iter=10, random_state=42)
clf_lr.fit(bag_of_words, training_set['subreddit'])
```

```
c:\users\rezam\appdata\local\programs\python\python38-32\lib\site-packa
ges\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

Out[22]:
```
LogisticRegression(max_iter=10, random_state=42)
```

Logistic Regression - Cross Validation:

In [23]:
```
clf_lr.score(bag_of_words, training_set['subreddit'])
```

Out[23]:
```
0.4988166666666667
```

Logistic Regression - Cross Validation:

In [25]:
```
scores_lr = cross_val_score(clf_lr, bag_of_words, training_set['subredd
it'], cv=5)
scores_lr
```

```
c:\users\rezam\appdata\local\programs\python\python38-32\lib\site-packa
ges\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\users\rezam\appdata\local\programs\python\python38-32\lib\site-packa
ges\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\users\rezam\appdata\local\programs\python\python38-32\lib\site-packa
ges\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\users\rezam\appdata\local\programs\python\python38-32\lib\site-packa
ges\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\users\rezam\appdata\local\programs\python\python38-32\lib\site-packa
ges\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs fa
iled to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
```

Out[25]:
```
array([0.41966667, 0.41683333, 0.42291667, 0.42075    , 0.42783333])
```

In [ ]: