```
!pip install PyDrive
```

```
Collecting PyDrive
  Downloading https://files.pythonhosted.org/packages/52/e0/0e64788e5dd58ce2d6934549676243dc69d982f198524be9b99e9c
     |████████████████████████████████| 993kB 2.1MB/s
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/python3.6/dist-packages (from PyDri
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (4.1.3
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (3.13)
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.6/dist-packages (from google-api-py
Requirement already satisfied: six<2dev,>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from google-api-python-
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from google-api
Requirement already satisfied: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.6/dist-packages (from oauth2client>
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->PyD
Building wheels for collected packages: PyDrive
  Building wheel for PyDrive (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/fa/d2/9a/d3b6b506c2da98289e5d417215ce34b696db856643bad779f4
Successfully built PyDrive
Installing collected packages: PyDrive
Successfully installed PyDrive-1.3.1
```

```
import os
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)
```

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

Saved successfully!

https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee

```
%cd "gdrive/My Drive/project"
```

```
/content/gdrive/My Drive/project
```

```
!ls
!python
```

```
data            embeddings.h5  n_words_clean.eps  rnn
data_loader.py  mltools        __pycache__        wiki.en.vec
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

Creating model and training model

Naive Bayes, Logistic Regression Basline models

```
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score, roc_auc_score
from scipy.sparse import hstack
from sklearn.pipeline import make_union
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
```

```
train, valid = data_loader.load_train_data('data/train.csv', valid_rate=0.1)
train = train.fillna('')
valid = valid.fillna('')
test = data_loader.load_test_data('data/test.csv','data/test_labels.csv').fillna('')
```

```
import re, string
re_tok = re.compile(f'([{string.punctuation}“”¨«»®´·º½¾¿¡§£₤‘’])')
def tokenize(s): return re_tok.sub(r' \1 ', s).split()
```

Saved successfully!

```
    "yaay": " good ",
    "yaaay": " good ",
    "yaaaay": " good ",
    "yaaaaay": " good ",
    ":/": " bad ",
    ":&gt;": " sad ",
    ":')": " sad ",
    ":-(": " frown ",
    ":(": " frown ",
    ":s": " frown ",
    ":-s": " frown ",
    "&lt;3": " heart ",
    ":d": " smile ",
    ":p": " smile ",
    ":dd": " smile ",
    "8)": " smile ",
    ":-)": " smile ",
    ":)": " smile ",
    ";)": " smile ",
    "(-:": " smile ",
    "(:": " smile ",
    ":/": " worry ",
    ":&gt;": " angry ",
    ":')": " sad ",
    ":-(": " sad ",
    ":(": " sad ",
    ":s": " sad ",
    ":-s": " sad ",
    r"\br\b": "are",
    r"\bu\b": "you",
    r"\bhaha\b": "ha",
    r"\bhahaha\b": "ha",
    r"\bdon't\b": "do not",
    r"\bdoesn't\b": "does not",
    r"\bdidn't\b": "did not",
    r"\bhasn't\b": "has not",
    r"\bhaven't\b": "have not",
    r"\bhadn't\b": "had not",
    r"\bwon't\b": "will not",
    r"\bwouldn't\b": "would not",
    r"\bcan't\b": "can not",
    r"\bcannot\b": "can not",
    r"\bi'm\b": "i am",
    "m": "am",
    "r": "are",
    "u": "you",
    "haha": "ha",
    "hahaha": "ha",
    "don't": "do not",
```

Saved successfully!

```
    "hasn't" : "has not",
    "haven't": "have not",
    "hadn't": "had not",
    "won't": "will not",
    "wouldn't": "would not",
    "can't": "can not",
    "cannot": "can not",
    "i'm": "i am",
    "m": "am",
    "i'll" : "i will",
    "its" : "it is",
    "it's" : "it is",
    "'s" : " is",
    "that's" : "that is",
    "weren't" : "were not",
}


new_train_data = []
new_test_data = []
new_valid_data = []

list_train = train['comment_text'].tolist()
list_test = test['comment_text'].tolist()
list_valid = valid['comment_text'].tolist()

for i in list_train:
    arr = str(i).split()
    xx = ""
    for j in arr:
        j = str(j).lower()
        if j[:4] == 'http' or j[:3] == 'www':
            continue
        if j in repl.keys():
            j = repl[j]
        xx = xx + j + " "
    new_train_data.append(xx)

for i in list_test:
    arr = str(i).split()
    xx = ""
    for j in arr:
        j = str(j).lower()
        if j[:4] == 'http' or j[:3] == 'www':
            continue
        if j in repl.keys():
            j = repl[j]
        xx = xx + j + " "
```

Saved successfully!

```
    arr = str(i).split()
    xx = ""
    for j in arr:
        j = str(j).lower()
        if j[:4] == 'http' or j[:3] == 'www':
            continue
        if j in repl.keys():
            j = repl[j]
        xx = xx + j + " "
    new_valid_data.append(xx)

train["clean_comment_text"] = new_train_data
test["clean_comment_text"] = new_test_data
valid["clean_comment_text"] = new_valid_data


pattern = re.compile(r'[^a-zA-Z ?!]+')
train_text = train["clean_comment_text"].tolist()
test_text = test["clean_comment_text"].tolist()
valid_text = valid["clean_comment_text"].tolist()
for i,c in enumerate(train_text):
    train_text[i] = pattern.sub('',train_text[i].lower())
for i,c in enumerate(test_text):
    test_text[i] = pattern.sub('',test_text[i].lower())
for i,c in enumerate(valid_text):
    valid_text[i] = pattern.sub('',valid_text[i].lower())


train['comment_text'] = train_text
test["comment_text"] = test_text
valid["comment_text"] = valid_text
del train_text, test_text, valid_text
train.drop(['clean_comment_text'], inplace = True, axis = 1)
test.drop(['clean_comment_text'], inplace = True, axis = 1)
valid.drop(['clean_comment_text'], inplace = True, axis = 1)
```

Word2Vec vectors sometimes contain negative values, whereas Naive Bayes is only compatible with positive values (it assumes document frequencies). Therefore, using TF-IDF vectorization. This way, results can be compared.

```
all_text = pd.concat([train['comment_text'],valid['comment_text'], test['comment_text']])

word_vectorizer = TfidfVectorizer(ngram_range =(1,3),
                            tokenizer=tokenize,
                            min_df=3, max_df=0.9,
                            strip_accents='unicode',
```

Saved successfully!

```
                                ords = 'english',
                                er = 'word',
                              use_idf=1,
                              smooth_idf=1,
                              sublinear_tf=1 )

char_vectorizer = TfidfVectorizer(ngram_range =(1,4),
                                  min_df=3, max_df=0.9,
                                  strip_accents='unicode',
                                  analyzer = 'char',
                                  stop_words = 'english',
                                  use_idf=1,
                                  smooth_idf=1,
                                  sublinear_tf=1,
                                  max_features=50000)

vectorizer = make_union(word_vectorizer, char_vectorizer)

vectorizer.fit(all_text)

train_matrix =vectorizer.transform(train['comment_text'])
test_matrix = vectorizer.transform(test['comment_text'])
valid_matrix = vectorizer.transform(valid['comment_text'])


test_score = []
val_score = []
def scoring_model(model, cl):
      model.fit(train_matrix, train[cl])
      pred_valid = model.predict(valid_matrix)
      pred_test = model.predict(test_matrix)
      score_valid = roc_auc_score(valid[cl], pred_valid)
      score_test = roc_auc_score(test[cl], pred_test)
      val_score.append(score_valid.mean())
      test_score.append(score_test.mean())
      print(cl)
      print(score_valid)
      print(score_test)


from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
from sklearn.linear_model import LogisticRegression
class_names = ['toxic','severe_toxic','obscene','threat','insult','identity_hate']


model = MultinomialNB()
for cl in class_names:
    scoring_model(model, cl)
```

Saved successfully!

```
0.6237755117961351
0.6291918554463587
severe_toxic
0.5
0.49996069862130765
obscene
0.5964362460019862
0.586382221818522
threat
0.5
0.5
insult
0.5682783987793957
0.5577463447397811
identity_hate
0.5
0.50209093529478
```

```
MNB_score_val = val_score
MNB_score_test = test_score
print(MNB_score_val)
print(MNB_score_test)
```

```
[0.6237755117961351, 0.5, 0.5964362460019862, 0.5, 0.5682783987793957, 0.5]
[0.6291918554463587, 0.49996069862130765, 0.586382221818522, 0.5, 0.5577463447397811, 0.50209093529478]
```

```
val_score = []
test_score = []
LR_model = LogisticRegression(C=3, dual=True)
for cl in class_names:
    scoring_model(LR_model,cl)
```

Saved successfully!

```
0.8587940071808446
severe_toxic
0.6365839877729541
0.6631916180589643
obscene
0.8570492914008244
0.8283497854705351
```

```
LR_score_val = val_score
LR_score_test = test_score
print(LR_score_val)
print(LR_score_test)
```

```
[0.8361058061822305, 0.6365839877729541, 0.8570492914008244, 0.624811106913487, 0.7826107402653089, 0.667325293346
[0.8587940071808446, 0.6631916180589643, 0.8283497854705351, 0.6490146629052437, 0.7802253834915703, 0.68317474071
```

```
def pr(y_i, y):
    p = train_matrix[y==y_i].sum(0)
    return (p+1) / ((y==y_i).sum()+1)


def get_mdl(y):
    y = y.values
    r = np.log(pr(1,y) / pr(0,y))
    m = LogisticRegression(C=3, dual=True)
    x_nb = train_matrix.multiply(r)
    return m.fit(x_nb, y), r


model = LogisticRegression(C=3,dual = True)
NBLR_score_val=[]
NBLR_score_test=[]
for cl in class_names:
    y = train[cl].values
    r = np.log(pr(1,y) / pr(0,y))
    x_nb = train_matrix.multiply(r)
    model.fit(x_nb, y)
    pred_valid = model.predict(valid_matrix)
    pred_test = model.predict(test_matrix)
    score_valid = roc_auc_score(valid[cl], pred_valid)
    score_test = roc_auc_score(test[cl], pred_test)
    print(cl)
    print(score_valid)
```

Saved successfully!

```
id.mean())
st.mean())
```

```
toxic
0.6673971959266566
0.6371747839211169
severe_toxic
0.5
0.5
obscene
0.5627810234251229
0.5549765304196039
threat
0.5
0.5
insult
0.5559448394902956
0.5381409147356565
identity_hate
0.5
0.5
```

```
DF_score = pd.DataFrame(index=class_names)
DF_score['MNB-valid'] = MNB_score_val
DF_score['MNB-test'] = MNB_score_test
DF_score['LR-valid'] = LR_score_val
DF_score['LR-test'] = LR_score_test
DF_score['NBLR-valid'] = NBLR_score_val
DF_score['NBLR-test'] = NBLR_score_test
print(DF_score)
```

```
                MNB-valid  MNB-test  LR-valid   LR-test  NBLR-valid  NBLR-test
toxic            0.623776  0.629192  0.836106  0.858794    0.667397   0.637175
severe_toxic     0.500000  0.499961  0.636584  0.663192    0.500000   0.500000
obscene          0.596436  0.586382  0.857049  0.828350    0.562781   0.554977
threat           0.500000  0.500000  0.624811  0.649015    0.500000   0.500000
insult           0.568278  0.557746  0.782611  0.780225    0.555945   0.538141
identity_hate    0.500000  0.502091  0.667325  0.683175    0.500000   0.500000
```

```
print("MNB validiation AUC ROC", sum(DF_score["MNB-valid"])/6)
print("MNB test AUC ROC", sum(DF_score["MNB-test"])/6)
print("LR validiation AUC ROC", sum(DF_score["LR-valid"])/6)
```

Saved successfully!

```
re["LR-test"])/6)
um(DF_score["NBLR-valid"])/6)
print( NBLR test AUC ROC , sum(DF_score["NBLR-test"])/6)


preds = np.zeros((len(test), len(class_names)))

for i, j in enumerate(class_names):
    m,r = get_mdl(train[j])
    preds[:,i] = m.predict_proba(test_matrix.multiply(r))[:,1]

np.save("nblr-svm/test_predict.npy", preds)


#Submission
subm = pd.read_csv('data/sample_submission.csv')
predictions = np.load('nblr-svm/test_predict.npy')
submid = pd.DataFrame({'id': subm["id"]})
submission = pd.concat([submid, pd.DataFrame(predictions, columns = label_cols)], axis=1)
submission.columns =  ['id','prediction']
submission.to_csv('nblr-svm/submission.csv', index=False)
```

Saved successfully!