**RAW CODE: -**

{

"cells": [

{

"cell\_type": "markdown",

"metadata": {

"colab\_type": "text",

"id": "kaYNMVqj5yxo"

},

"source": [

"## Preface\n",

"\n",

"This is an analysis of Wikipedia comments to create models that identify various types of toxic comments. There is a lot of racist content and swear words in the dataset and some of it will pop up in the analysis. "

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"metadata": {

"colab": {

"autoexec": {

"startup": false,

"wait\_interval": 0

},

"base\_uri": "https://localhost:8080/",

"height": 920,

"output\_extras": [

{

"item\_id": 23

}

]

},

"colab\_type": "code",

"executionInfo": {

"elapsed": 32492,

"status": "ok",

"timestamp": 1522886463906,

"user": {

"displayName": "Jay Speidell",

"photoUrl": "//lh6.googleusercontent.com/-PA283dAAXaQ/AAAAAAAAAAI/AAAAAAAAGng/nfaP1-4Q3qM/s50-c-k-no/photo.jpg",

"userId": "112011094715710279880"

},

"user\_tz": 420

},

"id": "It7csY1f5yxs",

"outputId": "a415ec57-e23e-4d58-b3bf-b46882562dbc"

},

"outputs": [],

"source": [

"import pandas as pd\n",

"import pickle\n",

"import numpy as np\n",

"import nltk\n",

"from nltk.corpus import stopwords\n",

"import keras\n",

"import time\n",

"import matplotlib.pyplot as plt\n",

"import seaborn as sns\n",

"import re\n",

"import string\n",

"from collections import namedtuple\n",

"from sklearn.preprocessing import StandardScaler\n",

"from sklearn.model\_selection import cross\_val\_score\n",

"from sklearn.model\_selection import StratifiedKFold\n",

"from sklearn.model\_selection import train\_test\_split \n",

"from sklearn.metrics import f1\_score\n",

"from sklearn.feature\_extraction.text import TfidfVectorizer\n",

"from scipy import sparse\n",

"from sklearn.pipeline import make\_union\n",

"from sklearn.svm import LinearSVC\n",

"from sklearn.naive\_bayes import MultinomialNB\n",

"from sklearn.linear\_model import LogisticRegression\n",

"import lightgbm as lgb\n",

"from sklearn.metrics import confusion\_matrix"

]

},

{

"cell\_type": "code",

"execution\_count": 8,

"metadata": {},

"outputs": [],

"source": [

"# Global random state and k-fold strategy \n",

"seed = 42\n",

"k = 5\n",

"cv = StratifiedKFold(n\_splits=k, random\_state=seed)"

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {},

"outputs": [],

"source": [

"def lgb\_f1\_score(y\_hat, data):\n",

" # https://stackoverflow.com/questions/49774825/python-lightgbm-cross-validation-how-to-use-lightgbm-cv-for-regression\n",

" y\_true = data.get\_label()\n",

" y\_hat = np.round(y\_hat) \n",

" return 'f1', f1\_score(y\_true, y\_hat), True"

]

},

{

"cell\_type": "code",

"execution\_count": 10,

"metadata": {

"colab": {

"autoexec": {

"startup": false,

"wait\_interval": 0

}

},

"colab\_type": "code",

"id": "keYiW4Ws5yx2"

},

"outputs": [],

"source": [

"start = time.time()\n",

"def print\_time(start):\n",

" time\_now = time.time() - start \n",

" minutes = int(time\_now / 60)\n",

" seconds = int(time\_now % 60)\n",

" if seconds < 10:\n",

" print('Elapsed time was %d:0%d.' % (minutes, seconds))\n",

" else:\n",

" print('Elapsed time was %d:%d.' % (minutes, seconds))"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Feature Engineering "

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"metadata": {},

"outputs": [],

"source": [

"def feature\_engineering(df, sparse=0): \n",

" \n",

" # Comment length\n",

" df['length'] = df.comment\_text.apply(lambda x: len(x))\n",

" \n",

"\n",

" # Capitalization percentage\n",

" def pct\_caps(s):\n",

" return sum([1 for c in s if c.isupper()]) / (sum(([1 for c in s if c.isalpha()])) + 1)\n",

" df['caps'] = df.comment\_text.apply(lambda x: pct\_caps(x))\n",

"\n",

" # Mean Word length \n",

" def word\_length(s):\n",

" s = s.split(' ')\n",

" return np.mean([len(w) for w in s if w.isalpha()])\n",

" df['word\_length'] = df.comment\_text.apply(lambda x: word\_length(x))\n",

"\n",

" # Average number of exclamation points \n",

" df['exclamation'] = df.comment\_text.apply(lambda s: len([c for c in s if c == '!']))\n",

"\n",

" # Average number of question marks \n",

" df['question'] = df.comment\_text.apply(lambda s: len([c for c in s if c == '?']))\n",

" \n",

" # Normalize\n",

" for label in ['length', 'caps', 'word\_length', 'question', 'exclamation']:\n",

" minimum = df[label].min()\n",

" diff = df[label].max() - minimum\n",

" df[label] = df[label].apply(lambda x: (x-minimum) / (diff))\n",

"\n",

" # Strip IP Addresses\n",

" ip = re.compile('(([2][5][0-5]\\.)|([2][0-4][0-9]\\.)|([0-1]?[0-9]?[0-9]\\.)){3}'\n",

" +'(([2][5][0-5])|([2][0-4][0-9])|([0-1]?[0-9]?[0-9]))')\n",

" def strip\_ip(s, ip):\n",

" try:\n",

" found = ip.search(s)\n",

" return s.replace(found.group(), ' ')\n",

" except:\n",

" return s\n",

"\n",

" df.comment\_text = df.comment\_text.apply(lambda x: strip\_ip(x, ip))\n",

" \n",

" return df\n",

"\n",

"def merge\_features(comment\_text, data, engineered\_features):\n",

" new\_features = sparse.csr\_matrix(df[engineered\_features].values)\n",

" if np.isnan(new\_features.data).any():\n",

" new\_features.data = np.nan\_to\_num(new\_features.data)\n",

" return sparse.hstack([comment\_text, new\_features])"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Loading Dataset"

]

},

{

"cell\_type": "code",

"execution\_count": 12,

"metadata": {},

"outputs": [

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/numpy/core/fromnumeric.py:2889: RuntimeWarning: Mean of empty slice.\n",

" out=out, \*\*kwargs)\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Training labels:\n",

"['toxic', 'severe\_toxic', 'obscene', 'threat', 'insult', 'identity\_hate']\n",

"(159571, 6)\n",

"\n",

"Training data\n",

"['comment\_text', 'length', 'caps', 'word\_length', 'exclamation', 'question']\n",

"(159571, 6)\n",

"\n",

"Submission data\n",

"['comment\_text', 'length', 'caps', 'word\_length', 'exclamation', 'question']\n",

"(153164, 6)\n",

"['length', 'caps', 'word\_length', 'exclamation', 'question']\n"

]

}

],

"source": [

"# Reset data and create holdout set. \n",

"\n",

"df = pd.read\_csv('train.csv')\n",

"targets = list(df.columns[2:])\n",

"df\_targets = df[targets].copy()\n",

"\n",

"df\_sub = pd.read\_csv('test.csv', dtype={'id': object}, na\_filter=False)\n",

"\n",

"submission = pd.DataFrame()\n",

"submission['id'] = df\_sub.id.copy()\n",

"\n",

"# Feature Engineering\n",

"df = feature\_engineering(df)\n",

"df\_sub = feature\_engineering(df\_sub)\n",

"\n",

"print('Training labels:')\n",

"print(list(df\_targets.columns))\n",

"print(df\_targets.shape)\n",

"\n",

"print('\\nTraining data')\n",

"df.drop(list(df\_targets.columns), inplace=True, axis=1)\n",

"df.drop('id', inplace=True, axis=1)\n",

"print(list(df.columns))\n",

"print(df.shape)\n",

"\n",

"\n",

"print('\\nSubmission data')\n",

"df\_sub.drop('id', inplace=True, axis=1)\n",

"print(list(df\_sub.columns))\n",

"print(df\_sub.shape)\n",

"\n",

"toxic\_rows = df\_targets.sum(axis=1)\n",

"toxic\_rows = (toxic\_rows > 0)\n",

"targets.append('any\_label')\n",

"df\_targets['any\_label'] = toxic\_rows.astype(int)\n",

"\n",

"new\_features = list(df.columns[1:])\n",

"print(new\_features)\n",

"\n",

"from sklearn.model\_selection import train\_test\_split\n",

"df, holdout, df\_targets, holdout\_targets = train\_test\_split(df, df\_targets, test\_size=0.2, random\_state=seed)"

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"['length', 'caps', 'word\_length', 'exclamation', 'question']"

]

},

"execution\_count": 13,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"new\_features"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Multilabel Function "

]

},

{

"cell\_type": "code",

"execution\_count": 14,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.base import clone\n",

"#todo \n",

"# Weights for \n",

"def multi\_cv(model, data, labels, k=5, nb\_features=False):\n",

" cv = StratifiedKFold(n\_splits=k, random\_state=seed)\n",

" # Creating NB features just once from any\_label has about the same \n",

" # performance as individual labels with faster speed. \n",

" def log\_count\_ratio(x, y):\n",

" x = sparse.csr\_matrix(x)\n",

" # WARNING: Some scipy modules use indexes that start at 1! \n",

" # You need to add 1 to an index when performing operations on a csr\_matrix \n",

"\n",

" p = abs(x[np.where(y==1)].sum(axis=0))\n",

" p = p + 1\n",

" p = p / np.sum(p)\n",

"\n",

" q = abs(x[np.where(y==0)].sum(axis=0))\n",

" q = q + 1\n",

" q = q / np.sum(q)\n",

"\n",

" return np.log(p/q)\n",

" \n",

" # Labels must be in a dataframe\n",

" scores = []\n",

" r\_values = []\n",

" for label in labels.columns:\n",

" if nb\_features:\n",

" r = log\_count\_ratio(data, labels[label])\n",

" r\_values.append(r)\n",

" data = data.multiply(r)\n",

" if np.isnan(data.data).any():\n",

" data.data = np.nan\_to\_num(data.data)\n",

" score = np.mean(cross\_val\_score(clone(model), data, labels[label], scoring='f1', cv=cv))\n",

" print(label + ' f1 score: %.4f' % score)\n",

" scores.append(score)\n",

" print('Average (excluding any) f1 score: %.4f' % np.mean(scores[:-1]))\n",

" if nb\_features:\n",

" return scores, r\_values\n",

" else:\n",

" return scores\n",

"\n",

"#training\_comments.data = np.nan\_to\_num(training\_comments.data)\n",

"\n",

"#model = LinearSVC()\n",

"#\_ = multi\_cv(model, training\_comments, df\_targets, nb\_features=True)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## NB Feature Transformer \n",

"\n",

"This is the primary method that I will use for the NB-SVM models, but I've left other code in to use as a reference. "

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"metadata": {},

"outputs": [],

"source": [

"class NBFeatures:\n",

" def \_\_init\_\_(self, epsilon=1, sparse=True):\n",

" # How much influence NB features have \n",

" if not epsilon > 0 and epsilon <= 1:\n",

" raise Exception(\"Invalid Epsilon value. Must be greater than zero and less than or equal to one.\")\n",

" self.epsilon = epsilon\n",

" self.r = None\n",

" \n",

" def log\_count\_ratio(self, x, y):\n",

" x = sparse.csr\_matrix(x)\n",

" # WARNING: Some scipy authors fall in the \"index starts at 1\" camp\n",

" # You need to add 1 to an index when performing operations on a csr\_matrix \n",

" p = abs(x[np.where(y==1)].sum(axis=0))\n",

" p = p + 1\n",

" p = p / np.sum(p)\n",

" q = abs(x[np.where(y==0)].sum(axis=0))\n",

" q = q + 1\n",

" q = q / np.sum(q)\n",

" return np.log(p/q)\n",

" \n",

" def fit(self, x, y):\n",

" self.r = self.log\_count\_ratio(x, y)\n",

" \n",

" def transform(self, x):\n",

" if self.r == None: \n",

" raise Exception(\"Model not fit, can't transform.\")\n",

" transformed = x.multiply(self.r)\n",

" return x.multiply(1-self.epsilon) + transformed.multiply(self.epsilon)\n",

" #return np.multiply(x, self.r)\n",

" \n",

" def fit\_transform(self, x, y):\n",

" self.r = self.log\_count\_ratio(x, y)\n",

" return self.transform(x, y)\n",

"\n",

"\n",

"#nb\_trans = NBFeatures(0.5)\n",

"#new = nb\_trans.fit\_transform(training\_comments, np.array(df\_targets.iloc[:,-1]))\n",

"#nb\_trans.r.shape"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"A separate helper function to calculate the log count ratio that can be used for experimentation. "

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {},

"outputs": [],

"source": [

"def log\_count\_ratio(x, y):\n",

" x = sparse.csr\_matrix(x)\n",

" # WARNING: Some scipy authors fall in the \"index starts at 1\" camp\n",

" # You need to add 1 to an index when performing operations on a csr\_matrix \n",

"\n",

" p = abs(x[np.where(y==1)].sum(axis=0))\n",

" p = p + 1\n",

" p = p / np.sum(p)\n",

"\n",

" q = abs(x[np.where(y==0)].sum(axis=0))\n",

" q = q + 1\n",

" q = q / np.sum(q)\n",

"\n",

" return np.log(p/q)\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# Vectorizing text"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"It's necessary to vectorize text before inputting into machine learning models. This is a process of translating string data into numerical data that the computer can better understand. Vectorized data is usually sparse, with an array where the features contain either word counts or another way of representing the occurance of characters or words in a string. This is done with a vectorizer object, which stores a dictionary of characters or words and their associated integer representation, along with relevant statistics if applicable. \n",

"\n",

"The strategy I'm going to use here is term frequency - inverse document frequency. This is a statistic that describes the usefulness of a string of characters by looking at the frequency that it occurs in an individual document (here, a single comment) and the inverse of its frequency in all of the documents in the dataset. \n",

"\n",

"That means that a word that is used frequently in a comment in this dataset, but that few comments in the dataset feature, is probably useful to the model. But a string that occurs in nearly every document is almost useless. "

]

},

{

"cell\_type": "code",

"execution\_count": 15,

"metadata": {

"colab": {

"autoexec": {

"startup": false,

"wait\_interval": 0

}

},

"colab\_type": "code",

"id": "TGoGjh6b5y0i"

},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 0:19.\n",

"(127656, 10000)\n"

]

}

],

"source": [

"start = time.time()\n",

"comment\_vector = TfidfVectorizer(max\_features=10000, analyzer='word', #ngram\_range=(2, 6), \n",

" stop\_words='english')\n",

"training\_comments = comment\_vector.fit\_transform(df.comment\_text)\n",

"holdout\_comments = comment\_vector.transform(holdout.comment\_text)\n",

"submission\_comments = comment\_vector.transform(df\_sub.comment\_text)\n",

"print\_time(start)\n",

"\n",

"print(training\_comments.shape)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"One of the most important parameters to tune in this problem is the number of features and the n\_gram range in the TF-IDF vectorizer, as well as choosing whether to analyze the sequences by characters or words. Analyzing by single words initially gives very poor performance, possibly because slang words and misspellings reduce the frequency of individual bad words. "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"This is a simple function to play with reducing class imbalance. "

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"metadata": {},

"outputs": [],

"source": [

"# This is just experimental to learn about the behavior of models with imbalanced classes. \n",

"\n",

"from numpy.random import sample\n",

"\n",

"def imbalance\_reduction(p, y):\n",

" \"\"\"\n",

" For multilabel problems, keeps all rows with a \n",

" positive label and returns p% of data where label is zero. \n",

" \"\"\"\n",

" #reduce y\n",

" y = np.sum(y, axis=1)\n",

" p = 1-p\n",

" keep\_index = sample(len(y))\n",

" keep\_index = keep\_index + y\n",

" keep\_index[keep\_index>=p] = 1\n",

" keep\_index[keep\_index<p] = 0\n",

"\n",

" return np.where(keep\_index==1)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# Benchmarks"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Logistic Regression"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"With 10,000 vectorized features, but without engineered features. "

]

},

{

"cell\_type": "code",

"execution\_count": 26,

"metadata": {

"scrolled": true

},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic score: 0.7192\n",

"severe\_toxic score: 0.3224\n",

"obscene score: 0.7452\n",

"threat score: 0.2069\n",

"insult score: 0.6277\n",

"identity\_hate score: 0.2772\n",

"any\_label score: 0.7299\n",

"Elapsed time was 0:43.\n"

]

}

],

"source": [

"\n",

"start = time.time()\n",

"for target in targets: \n",

" lr = LogisticRegression(random\_state=seed)\n",

" print(target + ' score: %.4f' % np.mean(cross\_val\_score(lr, training\_comments, df\_targets[target], scoring='f1', cv=cv)))\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"With engineered features added in. "

]

},

{

"cell\_type": "code",

"execution\_count": 28,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic score: 0.7235\n",

"severe\_toxic score: 0.3475\n",

"obscene score: 0.7440\n",

"threat score: 0.2029\n",

"insult score: 0.6274\n",

"identity\_hate score: 0.2768\n",

"any\_label score: 0.7328\n",

"Elapsed time was 0:54.\n"

]

}

],

"source": [

"start = time.time()\n",

"for target in targets: \n",

" lr = LogisticRegression(random\_state=seed)\n",

" print(target + ' score: %.4f' % np.mean(cross\_val\_score(lr, merge\_features(training\_comments, df, new\_features), df\_targets[target], scoring='f1', cv=cv)))\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Naive Bayes"

]

},

{

"cell\_type": "code",

"execution\_count": 34,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic f1 score: 0.6581\n",

"severe\_toxic f1 score: 0.1032\n",

"obscene f1 score: 0.6672\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n",

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n",

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n",

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"threat f1 score: 0.0000\n",

"insult f1 score: 0.5603\n",

"identity\_hate f1 score: 0.0418\n",

"any\_label f1 score: 0.6670\n",

"Average (excluding any) f1 score: 0.3384\n",

"Elapsed time was 0:02.\n"

]

}

],

"source": [

"start = time.time() \n",

"\n",

"model = MultinomialNB(alpha=1.0)\n",

"\_ = multi\_cv(model, training\_comments, df\_targets)\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"With engineered features. "

]

},

{

"cell\_type": "code",

"execution\_count": 35,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic f1 score: 0.6662\n",

"severe\_toxic f1 score: 0.0992\n",

"obscene f1 score: 0.6742\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n",

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n",

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n",

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.\n",

" 'precision', 'predicted', average, warn\_for)\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"threat f1 score: 0.0000\n",

"insult f1 score: 0.5708\n",

"identity\_hate f1 score: 0.0369\n",

"any\_label f1 score: 0.6734\n",

"Average (excluding any) f1 score: 0.3412\n",

"Elapsed time was 0:03.\n"

]

}

],

"source": [

"start = time.time() \n",

"\n",

"model = MultinomialNB(alpha=1.0)\n",

"\_ = multi\_cv(model, merge\_features(training\_comments, df, new\_features), df\_targets)\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Support Vector Machine "

]

},

{

"cell\_type": "code",

"execution\_count": 37,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic f1 score: 0.7551\n",

"severe\_toxic f1 score: 0.3382\n",

"obscene f1 score: 0.7806\n",

"threat f1 score: 0.3658\n",

"insult f1 score: 0.6643\n",

"identity\_hate f1 score: 0.3562\n",

"any\_label f1 score: 0.7703\n",

"Average (excluding any) f1 score: 0.5434\n",

"Elapsed time was 0:25.\n"

]

}

],

"source": [

"start = time.time()\n",

"model = LinearSVC(random\_state=seed)\n",

"\_ = multi\_cv(model, training\_comments, df\_targets)\n",

"print\_time(start)"

]

},

{

"cell\_type": "code",

"execution\_count": 38,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic f1 score: 0.7594\n",

"severe\_toxic f1 score: 0.3492\n",

"obscene f1 score: 0.7797\n",

"threat f1 score: 0.3624\n",

"insult f1 score: 0.6643\n",

"identity\_hate f1 score: 0.3548\n",

"any\_label f1 score: 0.7739\n",

"Average (excluding any) f1 score: 0.5450\n",

"Elapsed time was 0:37.\n"

]

}

],

"source": [

"start = time.time()\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\_ = multi\_cv(model, merge\_features(training\_comments, df, new\_features), df\_targets)\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Support Vector Machine with Naive Bayes Features"

]

},

{

"cell\_type": "code",

"execution\_count": 41,

"metadata": {},

"outputs": [],

"source": [

"nb = NBFeatures()\n",

"nb.fit(training\_comments, df\_targets.any\_label)\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Mini test: Does feature scaling make a difference? Support vector machines are particularly vulnerable to unbalanced features, and I want to check whether scaling after the added step of the Naive Bayes feature transformation makes a difference. "

]

},

{

"cell\_type": "code",

"execution\_count": 43,

"metadata": {},

"outputs": [

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic f1 score: 0.7636\n",

"severe\_toxic f1 score: 0.3483\n",

"obscene f1 score: 0.7863\n",

"threat f1 score: 0.3374\n",

"insult f1 score: 0.6707\n",

"identity\_hate f1 score: 0.3474\n",

"any\_label f1 score: 0.7804\n",

"Average (excluding any) f1 score: 0.5423\n",

"Elapsed time was 3:14.\n"

]

}

],

"source": [

"# Unscaled \n",

"start = time.time()\n",

"model = LinearSVC(random\_state=seed)\n",

"sc = StandardScaler(with\_mean=False)\n",

"\_ = multi\_cv(model, nb.transform(training\_comments), df\_targets) \n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"toxic f1 score: 0.7636\n",

"severe\_toxic f1 score: 0.3483\n",

"obscene f1 score: 0.7863\n",

"threat f1 score: 0.3374\n",

"insult f1 score: 0.6707\n",

"identity\_hate f1 score: 0.3474\n",

"any\_label f1 score: 0.7804\n",

"Average (excluding any) f1 score: 0.5423\n",

"Elapsed time was 3:14."

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# Scaled start = time.time()\n",

"model = LinearSVC(random\_state=seed)\n",

"sc = StandardScaler(with\_mean=False)\n",

"\_ = multi\_cv(model, sc.fit\_transform(nb.transform(training\_comments)), df\_targets)\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"toxic f1 score: 0.6450\n",

"severe\_toxic f1 score: 0.1992\n",

"obscene f1 score: 0.5849\n",

"threat f1 score: 0.2039\n",

"insult f1 score: 0.4803\n",

"identity\_hate f1 score: 0.1735\n",

"any\_label f1 score: 0.6640\n",

"Average (excluding any) f1 score: 0.3811\n",

"Elapsed time was 7:08."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Feature scaling produced worse scores and took longer. I expected this, but just wanted to do a sanity check to validate my assumptions. "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"With engineered features. "

]

},

{

"cell\_type": "code",

"execution\_count": 44,

"metadata": {},

"outputs": [

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"toxic f1 score: 0.7696\n",

"severe\_toxic f1 score: 0.3623\n",

"obscene f1 score: 0.7869\n",

"threat f1 score: 0.3408\n",

"insult f1 score: 0.6727\n",

"identity\_hate f1 score: 0.3488\n",

"any\_label f1 score: 0.7842\n",

"Average (excluding any) f1 score: 0.5468\n",

"Elapsed time was 3:37.\n"

]

}

],

"source": [

"start = time.time()\n",

"model = LinearSVC(random\_state=seed)\n",

"\_ = multi\_cv(model, nb\_eng.transform(merge\_features(training\_comments, df, new\_features)), df\_targets)\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### LightGBM "

]

},

{

"cell\_type": "code",

"execution\_count": 45,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": 49,

"metadata": {},

"outputs": [

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/lightgbm/engine.py:390: UserWarning: Found `n\_estimators` in params. Will use it instead of argument\n",

" warnings.warn(\"Found `{}` in params. Will use it instead of argument\".format(alias))\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 5:05.\n",

"Final CV F1 score is 0.7470\n"

]

}

],

"source": [

"start = time.time()\n",

"train\_data = lgb.Dataset(training\_comments, label=df\_targets.any\_label.values)\n",

"params = {\n",

" 'boosting\_type': 'gbdt',\n",

" 'objective': 'binary',\n",

" 'verbose': 1,\n",

" 'num\_leaves': 64,\n",

" 'n\_estimators': 500, \n",

" 'learning\_rate': 0.05, \n",

" 'max\_depth': 16,\n",

" 'n\_jobs': -1,\n",

" 'seed': seed\n",

"}\n",

"\n",

"cv\_results = lgb.cv(\n",

" params,\n",

" train\_data,\n",

" num\_boost\_round=100,\n",

" nfold=5,\n",

" metrics='mae',\n",

" #early\_stopping\_rounds=10,\n",

" feval=lgb\_f1\_score\n",

" )\n",

"print\_time(start)\n",

"\n",

"print(\"Final CV F1 score is %.4f\" % cv\_results['f1-mean'][-1])"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Elapsed time was 5:05.\n",

"Final CV F1 score is 0.7470"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"With engineered features. "

]

},

{

"cell\_type": "code",

"execution\_count": 51,

"metadata": {},

"outputs": [

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/lightgbm/engine.py:390: UserWarning: Found `n\_estimators` in params. Will use it instead of argument\n",

" warnings.warn(\"Found `{}` in params. Will use it instead of argument\".format(alias))\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 4:58.\n",

"Final CV F1 score is 0.7573\n"

]

}

],

"source": [

"start = time.time()\n",

"train\_data = lgb.Dataset(merge\_features(training\_comments, df, new\_features), label=df\_targets.any\_label.values)\n",

"params = {\n",

" 'boosting\_type': 'gbdt',\n",

" 'objective': 'binary',\n",

" 'verbose': 1,\n",

" 'num\_leaves': 64,\n",

" 'n\_estimators': 500, \n",

" 'learning\_rate': 0.05, \n",

" 'max\_depth': 16,\n",

" 'n\_jobs': -1,\n",

" 'seed': seed\n",

"}\n",

"\n",

"cv\_results = lgb.cv(\n",

" params,\n",

" train\_data,\n",

" num\_boost\_round=100,\n",

" nfold=5,\n",

" metrics='mae',\n",

" #early\_stopping\_rounds=10,\n",

" feval=lgb\_f1\_score\n",

" )\n",

"print\_time(start)\n",

"\n",

"print(\"Final CV F1 score is %.4f\" % cv\_results['f1-mean'][-1])"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Elapsed time was 5:04.\n",

"Final CV F1 score is 0.7573"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# Model Refinement"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Step 1: Optimize tf-idf max features "

]

},

{

"cell\_type": "code",

"execution\_count": 29,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 0:19.\n",

"(127656, 10000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.784174616909\n",

"Elapsed time was 0:33.\n"

]

}

],

"source": [

"start = time.time()\n",

"comment\_vector = TfidfVectorizer(max\_features=10000, analyzer='word', #ngram\_range=(3, 7), \n",

" stop\_words='english')\n",

"training\_comments = comment\_vector.fit\_transform(df.comment\_text)\n",

"holdout\_comments = comment\_vector.transform(holdout.comment\_text)\n",

"submission\_comments = comment\_vector.transform(df\_sub.comment\_text)\n",

"print\_time(start)\n",

"\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.784174616909\n",

"Elapsed time was 0:33."

]

},

{

"cell\_type": "code",

"execution\_count": 28,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 0:09.\n",

"(127656, 30000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.792595764561\n",

"Elapsed time was 0:21.\n"

]

}

],

"source": [

"start = time.time()\n",

"comment\_vector = TfidfVectorizer(max\_features=30000, analyzer='word', # ngram\_range=(3, 7), \n",

" stop\_words='english')\n",

"training\_comments = comment\_vector.fit\_transform(df.comment\_text)\n",

"#holdout\_comments = comment\_vector.transform(holdout.comment\_text)\n",

"#submission\_comments = comment\_vector.transform(df\_sub.comment\_text)\n",

"print\_time(start)\n",

"\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.792595764561\n",

"Elapsed time was 0:21."

]

},

{

"cell\_type": "code",

"execution\_count": 27,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 0:38.\n",

"(127656, 30000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.787710758788\n",

"Elapsed time was 0:25.\n"

]

}

],

"source": [

"start = time.time()\n",

"comment\_vector = TfidfVectorizer(max\_features=30000, analyzer='word', ngram\_range=(1,2), \n",

" stop\_words='english')\n",

"training\_comments = comment\_vector.fit\_transform(df.comment\_text)\n",

"#holdout\_comments = comment\_vector.transform(holdout.comment\_text)\n",

"#submission\_comments = comment\_vector.transform(df\_sub.comment\_text)\n",

"print\_time(start)\n",

"\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.787710758788\n",

"Elapsed time was 0:25."

]

},

{

"cell\_type": "code",

"execution\_count": 20,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 3:34.\n",

"(127656, 30000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.404212938192\n",

"Elapsed time was 0:31.\n"

]

}

],

"source": [

"start = time.time()\n",

"comment\_vector = TfidfVectorizer(max\_features=30000, analyzer='word', ngram\_range=(2,6), \n",

" stop\_words='english')\n",

"training\_comments = comment\_vector.fit\_transform(df.comment\_text)\n",

"#holdout\_comments = comment\_vector.transform(holdout.comment\_text)\n",

"#submission\_comments = comment\_vector.transform(df\_sub.comment\_text)\n",

"print\_time(start)\n",

"\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.404212938192\n",

"Elapsed time was 0:31."

]

},

{

"cell\_type": "code",

"execution\_count": 21,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 7:04.\n",

"(127656, 20000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.774235174963\n",

"Elapsed time was 3:09.\n"

]

}

],

"source": [

"start = time.time()\n",

"comment\_vector = TfidfVectorizer(max\_features=20000, analyzer='char', ngram\_range=(3, 7), \n",

" stop\_words='english')\n",

"training\_comments = comment\_vector.fit\_transform(df.comment\_text)\n",

"#holdout\_comments = comment\_vector.transform(holdout.comment\_text)\n",

"#submission\_comments = comment\_vector.transform(df\_sub.comment\_text)\n",

"print\_time(start)\n",

"\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.774235174963\n",

"Elapsed time was 3:09."

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 7:25.\n",

"(127656, 10000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.796185423779\n",

"Elapsed time was 2:56.\n"

]

}

],

"source": [

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=5000, analyzer='word',# ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=5000, analyzer='char', ngram\_range=(3, 7), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"training\_comments = vectorizer.fit\_transform(df.comment\_text)\n",

"print\_time(start)\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.796185423779\n",

"Elapsed time was 2:56."

]

},

{

"cell\_type": "code",

"execution\_count": 23,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 7:30.\n",

"(127656, 20000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.798518003383\n",

"Elapsed time was 2:07.\n"

]

}

],

"source": [

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=15000, analyzer='word', ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=5000, analyzer='char', ngram\_range=(3, 7), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"training\_comments = vectorizer.fit\_transform(df.comment\_text)\n",

"print\_time(start)\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.798518003383\n",

"Elapsed time was 2:07."

]

},

{

"cell\_type": "code",

"execution\_count": 24,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 7:29.\n",

"(127656, 30000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.800512138358\n",

"Elapsed time was 2:36.\n"

]

}

],

"source": [

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=20000, analyzer='word', ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=10000, analyzer='char', ngram\_range=(3, 7), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"training\_comments = vectorizer.fit\_transform(df.comment\_text)\n",

"print\_time(start)\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.800512138358\n",

"Elapsed time was 2:36."

]

},

{

"cell\_type": "code",

"execution\_count": 25,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 3:03.\n",

"(127656, 30000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"0.80146407813\n",

"Elapsed time was 2:48.\n"

]

}

],

"source": [

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=20000, analyzer='word', ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=10000, analyzer='char', ngram\_range=(3, 5), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"training\_comments = vectorizer.fit\_transform(df.comment\_text)\n",

"print\_time(start)\n",

"print(training\_comments.shape)\n",

"\n",

"start = time.time()\n",

"\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"training\_comments = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))\n",

"\n",

"model = LinearSVC(random\_state=seed)\n",

"\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1', cv=cv))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"0.80146407813\n",

"Elapsed time was 2:48."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Step 2: Optimize NB Feature Weight"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"There appears to be an issue with the consistency of scores here. The variation in training time suggests that the support vector machine algorithm is struggling with a too-large range of input values. Using the the last and best training\_comments from the cell above. "

]

},

{

"cell\_type": "code",

"execution\_count": 72,

"metadata": {

"scrolled": false

},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 2:51.\n",

"(127656, 30000)\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.100000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.100000 score: 0.7966\n",

"Elapsed time was 1:03.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.200000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.200000 score: 0.7989\n",

"Elapsed time was 1:05.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.300000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.300000 score: 0.8003\n",

"Elapsed time was 1:08.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.400000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.400000 score: 0.8001\n",

"Elapsed time was 1:17.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.500000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.500000 score: 0.8008\n",

"Elapsed time was 1:28.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.600000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.600000 score: 0.8013\n",

"Elapsed time was 1:47.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.700000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.700000 score: 0.8007\n",

"Elapsed time was 2:06.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.800000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.800000 score: 0.8020\n",

"Elapsed time was 2:19.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"For epsilon 0.900000\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Epsilon 0.900000 score: 0.8019\n",

"Elapsed time was 2:36.\n"

]

}

],

"source": [

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=20000, analyzer='word', ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=10000, analyzer='char', ngram\_range=(3, 5), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"training\_comments = vectorizer.fit\_transform(df.comment\_text)\n",

"print\_time(start)\n",

"print(training\_comments.shape)\n",

"\n",

"for i in range(1,10):\n",

" start = time.time()\n",

" sc = StandardScaler(with\_mean=False)\n",

" epsilon = i/10\n",

" print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')\n",

" print('For epsilon %f' % epsilon)\n",

" nb\_temp = NBFeatures(epsilon=epsilon)\n",

" nb\_temp.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

" input\_data = nb\_temp.transform(merge\_features(training\_comments, df, new\_features))\n",

" model = LinearSVC(random\_state=seed)\n",

" score = np.mean(cross\_val\_score(model, input\_data, df\_targets.any\_label, scoring='f1', cv=cv))\n",

" print('Epsilon %f score: %.4f' % (epsilon, score))\n",

" print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"Epsilon 0.100000 score: 0.7966\n",

"Elapsed time was 1:03.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.200000 score: 0.7989\n",

"Elapsed time was 1:05.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.300000 score: 0.8003\n",

"Elapsed time was 1:08.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.400000 score: 0.8001\n",

"Elapsed time was 1:17.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.500000 score: 0.8008\n",

"Elapsed time was 1:28.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.600000 score: 0.8013\n",

"Elapsed time was 1:47.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.700000 score: 0.8007\n",

"Elapsed time was 2:06.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.800000 score: 0.8020\n",

"Elapsed time was 2:19.\n",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",

"Epsilon 0.900000 score: 0.8019\n",

"Elapsed time was 2:36.\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Step 3: SVM Parameter Tuning "

]

},

{

"cell\_type": "code",

"execution\_count": 36,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 2:59.\n",

"(127656, 30000)\n"

]

},

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

}

],

"source": [

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=20000, analyzer='word', ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=10000, analyzer='char', ngram\_range=(3, 5), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"training\_comments = vectorizer.fit\_transform(df.comment\_text)\n",

"print\_time(start)\n",

"print(training\_comments.shape)\n",

"\n",

"# Reset NB feature transformer epsilon value\n",

"nb\_eng = NBFeatures()\n",

"nb\_eng.fit(merge\_features(training\_comments, df, new\_features), df\_targets.any\_label)\n",

"input\_data = nb\_eng.transform(merge\_features(training\_comments, df, new\_features))"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Test Gaussian Kernel"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"from sklearn.svm import SVC"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"start = time.time()\n",

"\n",

"model = SVC(random\_state=seed)\n",

"score = np.mean(cross\_val\_score(model, training\_comments, df\_targets.any\_label, scoring='f1'))\n",

"\n",

"print(score)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"#### Test C (L2 Penalization Coefficient) "

]

},

{

"cell\_type": "code",

"execution\_count": 32,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"C: 0\n",

"0.804947502199\n",

"Elapsed time was 1:07.\n",

"C: 0\n",

"0.805441509001\n",

"Elapsed time was 1:32.\n",

"C: 1\n",

"0.80322039905\n",

"Elapsed time was 2:15.\n"

]

}

],

"source": [

"params = [0.3, 0.5, 0.8]\n",

"\n",

"for C in params:\n",

" print(\"C: %.f\" % C)\n",

" start = time.time() \n",

" model = LinearSVC(random\_state=seed, C=C)\n",

" score = np.mean(cross\_val\_score(model, input\_data, df\_targets.any\_label, scoring='f1', cv=cv))\n",

" print(score)\n",

" print\_time(start)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# Model Evaluation "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"### Optimimum Model"

]

},

{

"cell\_type": "code",

"execution\_count": 21,

"metadata": {},

"outputs": [

{

"name": "stderr",

"output\_type": "stream",

"text": [

"/home/jay/anaconda3/envs/py35/lib/python3.5/site-packages/ipykernel/\_\_main\_\_.py:25: FutureWarning: comparison to `None` will result in an elementwise object comparison in the future.\n"

]

},

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Elapsed time was 3:29.\n",

"Elapsed time was 0:19.\n"

]

}

],

"source": [

"# TF-IDF Vectorization\n",

"start = time.time()\n",

"word\_vectorizer = TfidfVectorizer(max\_features=20000, analyzer='word', ngram\_range=(1, 2), \n",

" stop\_words='english')\n",

"char\_vectorizer = TfidfVectorizer(max\_features=10000, analyzer='char', ngram\_range=(3, 5), \n",

" stop\_words='english')\n",

"vectorizer = make\_union(word\_vectorizer, char\_vectorizer, n\_jobs=-1)\n",

"\n",

"# Fit to and transform input data\n",

"X\_train = vectorizer.fit\_transform(df.comment\_text)\n",

"X\_test = vectorizer.transform(holdout.comment\_text)\n",

"\n",

"# Name training target data\n",

"y\_train = df\_targets.any\_label\n",

"y\_test = holdout\_targets.any\_label\n",

"\n",

"# Create and fit NB Feature extractor \n",

"nb = NBFeatures()\n",

"nb.fit(X\_train, y\_train)\n",

"\n",

"# Tranform input data\n",

"X\_train = nb.transform(X\_train)\n",

"X\_test = nb.transform(X\_test)\n",

"\n",

"print\_time(start)\n",

"\n",

"# Define model and fit to data \n",

"start = time.time()\n",

"model = LinearSVC(random\_state=seed, C=0.5)\n",

"model.fit(X\_train, y\_train)\n",

"\n",

"print\_time(start)"

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"metadata": {},

"outputs": [],

"source": [

"y\_pred = model.predict(X\_test)"

]

},

{

"cell\_type": "code",

"execution\_count": 23,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"True Positives: 2386\n",

"False Positives: 282\n",

"True Negatives: 28389\n",

"False Negatives: 858\n",

"Precision: 0.8943\n",

"Recall: 0.7355\n",

"F1 Score: 0.8072\n",

"Total Accuracy: 0.96%\n"

]

}

],

"source": [

"tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()\n",

"\n",

"acc = (tp+tn)/(tn+fn+tp+fp)\n",

"print(\"True Positives: %d\" % tp)\n",

"print(\"False Positives: %d\" % fp)\n",

"print(\"True Negatives: %d\" % tn)\n",

"print(\"False Negatives: %d\" % fn)\n",

"print(\"Precision: %.4f\" % (tp/(tp+fp)))\n",

"print(\"Recall: %.4f\" % (tp/(tp+fn)))\n",

"print(\"F1 Score: %.4f\" % f1\_score(y\_test, y\_pred))\n",

"print(\"Total Accuracy: %.2f%%\" % acc)"

]

},

{

"cell\_type": "code",

"execution\_count": 24,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 720x720 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"cm = confusion\_matrix(y\_test, y\_pred)\n",

"def cm\_heatmap(arr, title):\n",

" \"\"\"Internal. Only called by scoring function.\"\"\"\n",

" plt.figure('cm\_heatmap', figsize=(10,10))\n",

" plt.title(title + ' confusion matrix')\n",

" sns.heatmap(arr, square=True, annot=True, cmap='YlOrRd', fmt='g', cbar=False)\n",

" plt.xlabel(\"Truth\")\n",

" plt.ylabel(\"Prediction\")\n",

" #sns.set(font\_scale=3)\n",

" plt.show()\n",

"cm\_heatmap(cm, \"Toxic Comments\")"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"accelerator": "GPU",

"colab": {

"collapsed\_sections": [

"kaYNMVqj5yxo",

"yH8NDq2D5yx9",

"jxMujLal5yyN",

"KmQxwcqp5yyu",

"\_A0tzkGe5yyz",

"41hcbO6m5yy8",

"qZU1wqtf5yy9",

"WPNxOBNp5yzJ",

"QeJWnVa15yzZ",

"ZSCnkLiC5y0A",

"krzeQWNb5y0h",

"X7Yxshbs5y02",

"1O2erv3G5y07",

"lw2pO8mo5y0-",

"aCDEfw7h5y1F",

"VJj2r6ky5y1O",

"5yiO9G4V5y1U",

"c\_NzviMo5y1t",

"fVAFqZCz5y2P",

"6dEs1c0H5y2r"

],

"default\_view": {},

"name": "ToxicComments.ipynb",

"provenance": [],

"version": "0.3.2",

"views": {}

},

"kernelspec": {

"display\_name": "Python [conda env:py35]",

"language": "python",

"name": "conda-env-py35-py"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.5.5"

}

},

"nbformat": 4,

"nbformat\_minor": 1

}