Project 2: Topic Classification

In this project, you'll work with text data from newsgroup posts on a variety of topics. You'll train classifiers to distinguish posts by topics inferred from the text. Whereas with digit classification, where each input is relatively dense (represented as a 28x28 matrix of pixels, many of which are non-zero), here each document is relatively sparse (represented as a bag-of-words). Only a few words of the total vocabulary are active in any given document. The assumption is that a label depends only on the count of words, not their order.

The `sklearn` documentation on feature extraction may be useful: http://scikit-learn.org/stable/modules/feature_extraction.html

Each problem can be addressed succinctly with the included packages -- please don't add any more. Grading will be based on writing clean, commented code, along with a few short answers.

As always, you're welcome to work on the project in groups and discuss ideas on Slack, but **please prepare your own write-up with your own code.**

```
# This tells matplotlib not to try opening a new window for each plot.
%matplotlib inline

# General libraries.
import re
import numpy as np
import matplotlib.pyplot as plt

# SK-learn libraries for learning.
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

# SK-learn libraries for evaluation.
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report

# SK-learn library for importing the newsgroup data.
from sklearn.datasets import fetch_20newsgroups

# SK-learn libraries for feature extraction from text.
from sklearn.feature_extraction.text import *

import nltk
```

Load the data, stripping out metadata so that only textual features will be used, and restricting documents to 4 specific topics. By default, newsgroups data is split into training and test sets, but here the test set gets further split into development and test sets. (If you remove the categories argument from the fetch function calls, you'd get documents from all 20 topics.)

```
categories = ['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train',
                                      remove=('headers', 'footers', 'quotes'),
                                      categories=categories)
newsgroups_test  = fetch_20newsgroups(subset='test',
                                      remove=('headers', 'footers', 'quotes'),
                                      categories=categories)

num_test = int(len(newsgroups_test.target) / 2)
test_data, test_labels   = newsgroups_test.data[num_test:], newsgroups_test.target[nu
dev_data, dev_labels     = newsgroups_test.data[:num_test], newsgroups_test.target[:n
train_data, train_labels = newsgroups_train.data, newsgroups_train.target

print('training label shape:', train_labels.shape)
print('dev label shape:',      dev_labels.shape)
print('test label shape:',     test_labels.shape)
print('labels names:',         newsgroups_train.target_names)
label_names = newsgroups_train.target_names
```

```
training label shape: (2034,)
dev label shape: (676,)
test label shape: (677,)
labels names: ['alt.atheism', 'comp.graphics', 'sci.space', 'talk.religion.misc'
```

Part 1:

For each of the first 5 training examples, print the text of the message along with the label.

```
def P1(num_examples=5):
    ### STUDENT START ###
    for example, label in zip(train_data[:num_examples], train_labels[:num_examples])
        print(f'{label_names[label]}:\n{example}\n\n===============\n==============
    ### STUDENT END ###

P1(5)
```

```
JG>Sorry, _perijoves_...I'm not used to talking this language.

Couldn't we just say periapsis or apoapsis?
```

```
================
================


alt.atheism:
I have a request for those who would like to see Charley Wingate
respond to the "Charley Challenges" (and judging from my e-mail, there
appear to be quite a few of you.)

It is clear that Mr. Wingate intends to continue to post tangential or
unrelated articles while ingoring the Challenges themselves.  Between
the last two re-postings of the Challenges, I noted perhaps a dozen or
more posts by Mr. Wingate, none of which answered a single Challenge.


It seems unmistakable to me that Mr. Wingate hopes that the questions
will just go away, and he is doing his level best to change the
subject.  Given that this seems a rather common net.theist tactic, I
would like to suggest that we impress upon him our desire for answers,
in the following manner:

1. Ignore any future articles by Mr. Wingate that do not address the
Challenges, until he answers them or explictly announces that he
refuses to do so.

--or--

2. If you must respond to one of his articles, include within it
something similar to the following:

    "Please answer the questions posed to you in the Charley Challenges."

Really, I'm not looking to humiliate anyone here, I just want some
honest answers.  You wouldn't think that honesty would be too much to
ask from a devout Christian, would you?

Nevermind, that was a rhetorical question.

================
================


sci.space:
AW&ST  had a brief blurb on a Manned Lunar Exploration confernce
May 7th  at Crystal City Virginia, under the auspices of AIAA.

Does anyone know more about this?  How much, to attend????

Anyone want to go?

================
================
```

Part 2:

Transform the training data into a matrix of **word** unigram feature vectors. What is the size of the vocabulary? What is the average number of non-zero features per example? What is the fraction of the non-zero entries in the matrix? What are the 0th and last feature strings (in alphabetical order)? *Use `CountVectorization` and its `.fit_transform` method. Use `.nnz` and `.shape` attributes, and `.get_feature_names` method.*

Now transform the training data into a matrix of **word** unigram feature vectors using your own vocabulary with these 4 words: ["atheism", "graphics", "space", "religion"]. Confirm the size of the vocabulary. What is the average number of non-zero features per example? *Use `CountVectorization(vocabulary=...)` and its `.transform` method.*

Now transform the training data into a matrix of **character** bigram and trigram feature vectors. What is the size of the vocabulary? *Use `CountVectorization(analyzer=..., ngram_range=...)` and its `.fit_transform` method.*

Now transform the training data into a matrix of **word** unigram feature vectors and prune words that appear in fewer than 10 documents. What is the size of the vocabulary? *Use `CountVectorization(min_df=...)` and its `.fit_transform` method.*

Now again transform the training data into a matrix of **word** unigram feature vectors. What is the fraction of words in the development vocabulary that is missing from the training vocabulary? *Hint: Build vocabularies for both train and dev and look at the size of the difference.*

Notes:

- `.fit_transform` makes 2 passes through the data: first it computes the vocabulary ("fit"), second it converts the raw text into feature vectors using the vocabulary ("transform").
- `.fit_transform` and `.transform` return sparse matrix objects. See about them at http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.csr_matrix.html.

```
def P2():
    ### STUDENT START ###
    vec = CountVectorizer()
    X = vec.fit_transform(train_data, train_labels)
    features = vec.get_feature_names()
    features.sort()

    print('Making basic vectorizer...')

    print(f'\tIn the training data, the total vocabulary is {len(vec.get_feature_name
    print(f'\tThe average non-zero matrix entries per sample is {sum(x.nnz for x in X
    print(f'\tThe fraction of non-zero / total matrix items is {X.nnz} / {X.shape[0]
    print(f'\tThe 0th feature is "{features[0]}", and the last is "{features[-1]}"')

    print()
```

```
    print('Making vectorizer with limited vocabulary...')

    vec = CountVectorizer(vocabulary=['atheism', 'graphics', 'space', 'religion'])
    X = vec.transform(train_data)
    print(f'\tThe vocabulary of the new dataset is {vec.get_feature_names()}')
    print(f'\tThe average number of nonzero features per sample is {sum(x.nnz for x i

    print()
    print('Making vectorizer for character [2, 3]-grams...')

    vec = CountVectorizer(analyzer='char', ngram_range=(2, 3))
    X = vec.fit_transform(train_data, train_labels)
    print(f'\tThe vocabulary size of the new dataset is {len(vec.get_feature_names())

    print()
    print('Making vectorizer that prunes words showing up in <10 documents...')

    vec = CountVectorizer(min_df=10)
    X = vec.fit_transform(train_data, train_labels)
    print(f'\tThe vocabulary size of the new dataset is {len(vec.get_feature_names())

    print()
    print('Checking for words in the dev dataset missing in the training dataset...')
    train_features = set(features)
    vec = CountVectorizer()
    vec.fit_transform(dev_data, dev_labels)
    dev_features = set(vec.get_feature_names())

    dev_minus_training = dev_features - train_features
    print(f'\tOf the {len(dev_features)} words in the dev vocab, {len(dev_minus_train
    ### STUDENT END ###

P2()
```

```
Making basic vectorizer...
        In the training data, the total vocabulary is 26879 words
        The average non-zero matrix entries per sample is 96.70599803343165
        The fraction of non-zero / total matrix items is 196700 / 54671886, or 0
        The 0th feature is "00", and the last is "zyxel"

Making vectorizer with limited vocabulary...
        The vocabulary of the new dataset is ['atheism', 'graphics', 'space', 'r
        The average number of nonzero features per sample is 0.26843657817109146

Making vectorizer for character [2, 3]-grams...
        The vocabulary size of the new dataset is 35478

Making vectorizer that prunes words showing up in <10 documents...
        The vocabulary size of the new dataset is 3064

Checking for words in the dev dataset missing in the training dataset...
        Of the 16246 words in the dev vocab, 4027 are not in the training set.
```

Part 3:

Transform the training and development data to matrices of word unigram feature vectors.

1. Produce several k-Nearest Neigbors models by varying k, including one with k set to optimize f1 score. For each model, show the k value and f1 score.
2. Produce several Naive Bayes models by varying smoothing (alpha), including one with alpha set approximately to optimize f1 score. For each model, show the alpha value and f1 score.
3. Produce several Logistic Regression models by varying L2 regularization strength (C), including one with C set approximately to optimize f1 score. For each model, show the C value, f1 score, and sum of squared weights for each topic.

- Why doesn't k-Nearest Neighbors work well for this problem?
- Why doesn't Logistic Regression work as well as Naive Bayes does?
- What is the relationship between logistic regression's sum of squared weights vs. C value?

Notes:

- Train on the transformed training data.
- Evaluate on the transformed development data.
- You can use `CountVectorizer` and its `.fit_transform` and `.transform` methods to transform data.
- You can use `KNeighborsClassifier(...)` to produce a k-Nearest Neighbors model.
- You can use `MultinomialNB(...)` to produce a Naive Bayes model.
- You can use `LogisticRegression(C=..., solver="liblinear", multi_class="auto")` to produce a Logistic Regression model.
- You can use `LogisticRegression`'s `.coef_` method to get weights for each topic.
- You can use `metrics.f1_score(..., average="weighted")` to compute f1 score.

```
def P3():
    ### STUDENT START ###
    # These train_X and dev_X are reused for each group
    train_vec = CountVectorizer()
    train_X = train_vec.fit_transform(train_data, train_labels)
    dev_X = train_vec.transform(dev_data)

    # KNN
    print('K Nearest Neighbors')
    for k in range(1, 12, 2):
        knn = KNeighborsClassifier(k)
        knn.fit(train_X, train_labels)

        predicted_labels = knn.predict(dev_X)
        f1_score = metrics.f1_score(dev_labels, predicted_labels, average='weighted')
```

```
        print(f'\tk = {k}:\tf-1 score: {f1_score}')

    print()
    print('K Nearest Neighbors doesn\'t work well for this problem because of the num
    print('KNN performs better with fewer dimensions and with more clustered data, wh
    print('has most of its values on the periphery of all the dimensions.')

    print()

    # Naive Bayes
    print('Naive Bayes')
    for exp in range(-3, 4):
        alpha = 10 ** exp
        mnb = MultinomialNB(alpha=alpha)
        mnb.fit(train_X, train_labels)

        predicted_labels = mnb.predict(dev_X)
        f1_score = metrics.f1_score(dev_labels, predicted_labels, average='weighted')
        print(f'\tα = {alpha}:  \tf-1 score: {f1_score}')

    print()

    # Logistic Regression
    print('Logistic Regression')
    for exp in range(-3, 4):
        c = 10 ** exp
        lr = LogisticRegression(C=c, solver='liblinear', multi_class='auto')
        lr.fit(train_X, train_labels)

        predicted_labels = lr.predict(dev_X)
        f1_score = metrics.f1_score(dev_labels, predicted_labels, average='weighted')
        print(f'\tC = {c}:  \tf-1 score: {f1_score}')
        print('\t\tSum of squared weights:\t' + ',\t'.join(f'{label_names[i]}: {sum(w

    print()

    print('Linear Regression performed worse than Naive Bayes because the labels for
    print('numeric. NB is able to learn categorical relationships between features th

    print()

    print('The higher the C value, the higher the sum of squared weights for each cat
    print('the sum of squared weights increases roughly linearly with C for the value

    ### STUDENT END ###

P3()
```

```
 K Nearest Neighbors
        k = 1:  f-1 score: 0.3805030018531525
        k = 3:  f-1 score: 0.4084150225437623
        k = 5:  f-1 score: 0.4287607236218357
```

```
        k = 7:  f-1 score: 0.45047910006117586
        k = 9:  f-1 score: 0.4365666176198027
        k = 11: f-1 score: 0.4266108018696209

K Nearest Neighbors doesn't work well for this problem because of the number of
KNN performs better with fewer dimensions and with more clustered data, while th
has most of its values on the periphery of all the dimensions.

Naive Bayes
        α = 0.001:      f-1 score: 0.7702518836155706
        α = 0.01:       f-1 score: 0.7751663218544357
        α = 0.1:        f-1 score: 0.7903052385098862
        α = 1:          f-1 score: 0.7777320236017224
        α = 10:         f-1 score: 0.6674814338256576
        α = 100:        f-1 score: 0.5100896536573467
        α = 1000:       f-1 score: 0.3146996158261945

Logistic Regression
        C = 0.001:      f-1 score: 0.6193046812006844
                Sum of squared weights: alt.atheism: 0.16509,   comp.graphics: 0
        C = 0.01:       f-1 score: 0.6646997417582748
                Sum of squared weights: alt.atheism: 2.5415,    comp.graphics: 2
        C = 0.1:        f-1 score: 0.6966243542418833
                Sum of squared weights: alt.atheism: 27.132,    comp.graphics: 2
        C = 1:          f-1 score: 0.6944172871853819
                Sum of squared weights: alt.atheism: 166.98,    comp.graphics: 1
        C = 10:         f-1 score: 0.6865669233056786
                Sum of squared weights: alt.atheism: 585.26,    comp.graphics: 4
        C = 100:        f-1 score: 0.6823892102438561
                Sum of squared weights: alt.atheism: 1409.4,    comp.graphics: 1
        C = 1000:       f-1 score: 0.6787813199733858
                Sum of squared weights: alt.atheism: 1913.5,    comp.graphics: 1

Linear Regression performed worse than Naive Bayes because the labels for this d
numeric. NB is able to learn categorical relationships between features that Lin

The higher the C value, the higher the sum of squared weights for each category.
the sum of squared weights increases roughly linearly with C for the values test
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarn
  "the number of iterations.", ConvergenceWarning)
```

ANSWER:

Part 4:

Transform the data to a matrix of word **bigram** feature vectors. Produce a Logistic Regression model. For each topic, find the 5 features with the largest weights (that's 20 features in total). Show a 20 row (features) x 4 column (topics) table of the weights.

Do you see any surprising features in this table?

Notes:

- Train on the transformed training data.
- You can use `CountVectorizer` and its `.fit_transform` method to transform data.
- You can use `LogisticRegression(C=0.5, solver="liblinear", multi_class="auto")` to produce a Logistic Regression model.
- You can use `LogisticRegression`'s `.coef_` method to get weights for each topic.
- You can use `np.argsort` to get indices sorted by element value.

```
def P4():
    ### STUDENT START ###
    # First, figure out the vocab and train a LR on it
    vec = CountVectorizer(ngram_range=(2, 2))
    X = vec.fit_transform(train_data, train_labels)
    vocab = vec.get_feature_names()

    lr = LogisticRegression(C=0.5, solver='liblinear', multi_class='auto')
    lr.fit(X, train_labels)

    # Now, find the most significant weights
    sorted_weights = lr.coef_.argsort()

    # This next code just organizes printing into nice columns
    feature_highlights = [[], [], [], []]
    for i, weights in enumerate(sorted_weights):
        feature_highlights[i] += [vocab[w] for w in weights[:-21:-1]]

    longest_flen = len(max((max(features, key=len) for features in feature_highlights
    col_len = longest_flen + 1

    for label in label_names:
        print(label.ljust(col_len), end='')
    print()
    print('=' * col_len * 4)

    # Print the discovered columns
    for row in zip(*feature_highlights):
        print(''.join(col.ljust(col_len) for col in row))

    print()
    print()
    print('In this table, I noticed that "cheers kent" was the second strongest alt.a
    print('Looking in the dataset, it looks like Kent is very active in the alt.athei
    print('Picking up on that might be overfitting, and might need manual tweaking to
    print('Beside that, I noticed that alt.atheism had more varied ideological discus
    print('talk.religion.misc was mostly focused on Christianity, and the tone of rel
    print('was a strong indicator between religion and atheism. The space and compute
    print('newsgroups mostly stayed on topic, and their key indicators were domain-sp
    print()
    print('One thing I noticed in all of them are that there are seemingly low-inform
```

```
    print('words, like "you are", "to my", "one of", or "out the".')

    ### STUDENT END ###

P4()
```

```
alt.atheism         comp.graphics       sci.space           talk.religion.misc
=====================================================================================
claim that          looking for         the space           the fbi
cheers kent         in advance          the moon            cheers kent
was just            comp graphics       sci space           ignorance is
you are             out there           and such            but he
are you             is there            it was              of jesus
in this             the image           the shuttle         off of
the faq             thanks in           space station       is strength
is not              know of             one of              the lord
you ve              any help            of space            the word
the motto           file is             in space            you were
to say              to my               why not             with you
you don            the screen          rather than         be with
look up             version of          used to             such lunacy
of islam            use the             nasa gov            teachings of
notion of           24 bit              to see              the teachings
of religion         does anyone         few years           compuserve com
re right            and it              sherzer methodology may be
natural morality    computer graphics   the spacecraft      out the
an atheist          how to              sounds like         how about
bake timmons        would like          the sun             objective morality


In this table, I noticed that "cheers kent" was the second strongest alt.atheism
Looking in the dataset, it looks like Kent is very active in the alt.atheism new
Picking up on that might be overfitting, and might need manual tweaking to fix.
Beside that, I noticed that alt.atheism had more varied ideological discussion,
talk.religion.misc was mostly focused on Christianity, and the tone of religious
was a strong indicator between religion and atheism. The space and computer grap
newsgroups mostly stayed on topic, and their key indicators were domain-specific

One thing I noticed in all of them are that there are seemingly low-information
words, like "you are", "to my", "one of", or "out the".
```

ANSWER:

Part 5:

To improve generalization, it is common to try preprocessing text in various ways before splitting into words. For example, you could try transforming strings to lower case, replacing sequences of numbers with single tokens, removing various non-letter characters, and shortening long words.

Produce a Logistic Regression model (with no preprocessing of text). Evaluate and show its f1 score and size of the dictionary.

Produce an improved Logistic Regression model by preprocessing the text. Evaluate and show its f1 score and size of the vocabulary. Try for an improvement in f1 score of at least 0.02.

How much did the improved model reduce the vocabulary size?

Notes:

- Train on the transformed training data.
- Evaluate on the transformed development data.
- You can use `CountVectorizer(preprocessor=...)` to preprocess strings with your own custom-defined function.
- `CountVectorizer` default is to preprocess strings to lower case.
- You can use `LogisticRegression(C=0.5, solver="liblinear", multi_class="auto")` to produce a logistic regression model.
- You can use `metrics.f1_score(..., average="weighted")` to compute f1 score.
- If you're not already familiar with regular expressions for manipulating strings, see https://docs.python.org/2/library/re.html, and re.sub() in particular.

```
import re

def better_preprocessor(s):
    ### STUDENT START ###
    import re

    s = ' ' + s.lower() + ' '  # Add whitespace on either end to make s easier to wor
    s = re.sub(r"(\w)'(\w)", r'\1\2', s)  # Replace quotes in contractions while leav
    s = re.sub(r'\d+', 'INT', s)  # Replace strings of digits with a single digit
    s = re.sub('INT.INT', 'FLOAT', s)  # Keeps decimal point numbers marked
    s = re.sub(r'\s(of|to|in|it|is|as|the)(?=\s)', '', s)  # Remove common, low-infor
    s = re.sub(r'([\.\\\-\/=_]{1,3})\1+', r' REP\1 ', s)  # Remove common repetitions
    s = re.sub('!+', '!', s)  # TONE!!!!! down!!!!! the! excitement.
    s = re.sub(r'\s(very|extremely)\s', ' ', s)  # Remove some filler words
    s = s.replace('ation', '')  # The model seemingly dislikes "ation"
    s = re.sub(r'(\w)([\.!?])\s', r'\1 \2 ', s)  # Split words into word-ending pairs

    return s.strip()
    ### STUDENT END ###

def P5():
    ### STUDENT START ###
    # Train a basic vectorizer
    vec = CountVectorizer()
    X = vec.fit_transform(train_data, train_labels)
    dev_X = vec.transform(dev_data)

    # Train a vectorizer with my custom preprocessor
    custom_vec = CountVectorizer(preprocessor=better_preprocessor)
```

```
    custom_X = custom_vec.fit_transform(train_data, train_labels)
    custom_dev_X = custom_vec.transform(dev_data)

    # Get predictions with it and show its performance
    lr = LogisticRegression(C=0.5, solver='liblinear', multi_class='auto')
    lr.fit(X, train_labels)
    pred = lr.predict(dev_X)

    custom_lr = LogisticRegression(C=0.5, solver='liblinear', multi_class='auto')
    custom_lr.fit(custom_X, train_labels)
    custom_pred = custom_lr.predict(custom_dev_X)

    print(f'Basic fitness:  {metrics.f1_score(dev_labels, pred, average="weighted")}
    print(f'Custom fitness: {metrics.f1_score(dev_labels, custom_pred, average="weigh
    print(f'The vocabulary changed by {len(custom_vec.get_feature_names()) - len(vec.
    ### STUDENT END ###

P5()
```

```
Basic fitness:  0.7084739776490449      vocab size: 26879
Custom fitness: 0.7339321018867501      vocab size: 24981
The vocabulary changed by -1898 words
```

Part 6:

The idea of regularization is to avoid learning very large weights (which are likely to fit the training data, but not generalize well) by adding a penalty to the total size of the learned weights. Logistic regression seeks the set of weights that minimizes errors in the training data AND has a small total size. The default L2 regularization computes this size as the sum of the squared weights (as in Part 3 above). L1 regularization computes this size as the sum of the absolute values of the weights. Whereas L2 regularization makes all the weights relatively small, L1 regularization drives many of the weights to 0, effectively removing unimportant features.

For several L1 regularization strengths ...

- Produce a Logistic Regression model using the **L1** regularization strength. Reduce the vocabulary to only those features that have at least one non-zero weight among the four categories. Produce a new Logistic Regression model using the reduced vocabulary and **L2** regularization strength of 0.5. Evaluate and show the L1 regularization strength, vocabulary size, and f1 score associated with the new model.

Show a plot of f1 score vs. log vocabulary size. Each point corresponds to a specific L1 regularization strength used to reduce the vocabulary.

How does performance of the models based on reduced vocabularies compare to that of a model based on the full vocabulary?

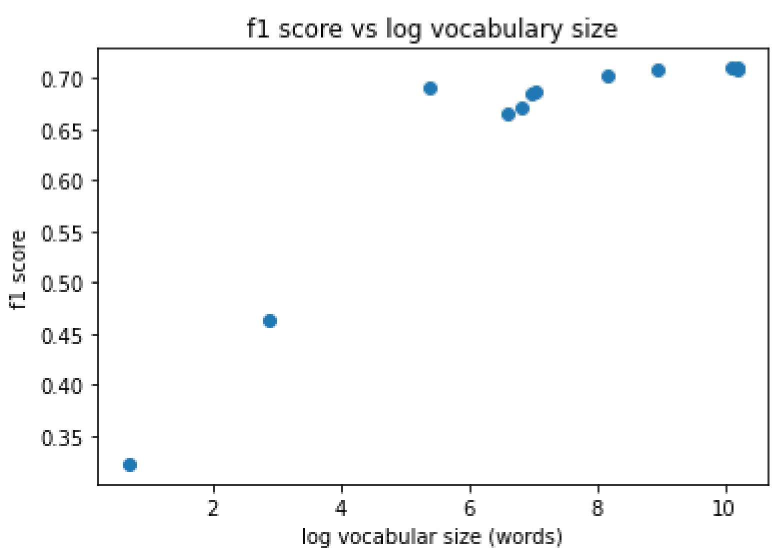
Notes:

- Train on the transformed training data.
- Evaluate on the transformed development data.
- You can use `LogisticRegression(..., penalty="l1")` to produce a logistic regression model using L1 regularization.
- You can use `LogisticRegression(..., penalty="l2")` to produce a logistic regression model using L2 regularization.
- You can use `LogisticRegression(..., tol=0.015)` to produce a logistic regression model using relaxed gradient descent convergence criteria. The gradient descent code that trains the logistic regression model sometimes has trouble converging with extreme settings of the C parameter. Relax the convergence criteria by setting tol=.015 (the default is .0001).

```
def P6():
    # Keep this random seed here to make comparison easier.
    np.random.seed(0)

    ### STUDENT START ###
    import math

    vec = CountVectorizer()
    X = vec.fit_transform(train_data, train_labels)
    labels = vec.get_feature_names()

    # Helper function that returns just the needed vocab
    def get_vocab(labels, weights):
        return [l for l, w in zip(labels, zip(*weights)) if sum(map(abs, w)) > 0]

    x = []
    y = []

    # Iterate over some pre-decided L1 penalties
    for c in [0.001, 0.01, 0.1, 0.5, 0.7, 0.9, 1, 10, 100, 1000, 10000, 100000]:
        # Train the L1 model and get its vocab
        lr = LogisticRegression(C=c, solver="liblinear", multi_class="auto", tol=0.01
        lr.fit(X, train_labels)

        vocab = get_vocab(labels, lr.coef_)

        # Sometimes the vocab is empty; skip in this case
        if len(vocab) == 0:
            print(f'For C={c}, the vocabulary was empty')
            continue

        # Make and test an L2 model with the chosen vocabulary
        smaller_vec = CountVectorizer(vocabulary=vocab)
        smaller_X = smaller_vec.transform(train_data)
        dev_X = smaller_vec.transform(dev_data)
```

```
        lr = LogisticRegression(C=0.5, solver='liblinear', multi_class='auto', penalt
        lr.fit(smaller_X, train_labels)
        pred = lr.predict(dev_X)

        f1 = metrics.f1_score(dev_labels, pred, average='weighted')
        print(f'For L1 C={c}, we found a vocab with {len(vocab)} words, with an f1 sc
        x.append(math.log(len(vocab)))
        y.append(f1)

    # Draws the scatter plot
    plt.scatter(x, y)
    plt.title('f1 score vs log vocabulary size')
    plt.xlabel('log vocabular size (words)')
    plt.ylabel('f1 score')

    ### STUDENT END ###

P6()
```

```
For L1 C=0.001, we found a vocab with 2 words, with an f1 score of 0.32250859062
For L1 C=0.01, we found a vocab with 18 words, with an f1 score of 0.46409420317
For L1 C=0.1, we found a vocab with 221 words, with an f1 score of 0.69006624486
For L1 C=0.5, we found a vocab with 739 words, with an f1 score of 0.66628919831
For L1 C=0.7, we found a vocab with 931 words, with an f1 score of 0.67188206227
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:947: ConvergenceWarn
  "the number of iterations.", ConvergenceWarning)
For L1 C=0.9, we found a vocab with 1081 words, with an f1 score of 0.6852508594
For L1 C=1, we found a vocab with 1163 words, with an f1 score of 0.687760059088
For L1 C=10, we found a vocab with 3576 words, with an f1 score of 0.70194877521
For L1 C=100, we found a vocab with 7706 words, with an f1 score of 0.7090742356
For L1 C=1000, we found a vocab with 24780 words, with an f1 score of 0.71007296
For L1 C=10000, we found a vocab with 26854 words, with an f1 score of 0.7100729
For L1 C=100000, we found a vocab with 26877 words, with an f1 score of 0.708473
```



ANSWER:

Part 7:

How is `TfidfVectorizer` different than `CountVectorizer`?

Produce a Logistic Regression model based on data represented in tf-idf form, with L2 regularization strength of 100. Evaluate and show the f1 score. How is `TfidfVectorizer` different than `CountVectorizer`?

Show the 3 documents with highest R ratio, where ...

$$R\,ratio = maximum\,predicted\,probability \div predicted\,probability\,of\,correct\,label$$

Explain what the R ratio describes. What kinds of mistakes is the model making? Suggest a way to address one particular issue that you see.

Note:

- Train on the transformed training data.
- Evaluate on the transformed development data.
- You can use `TfidfVectorizer` and its `.fit_transform` method to transform data to tf-idf form.
- You can use `LogisticRegression(C=100, solver="liblinear", multi_class="auto")` to produce a logistic regression model.
- You can use `LogisticRegression`'s `.predict_proba` method to access predicted probabilities.

```
def P7():
    ### STUDENT START ###
    print('TfidfVectorizer uses TF-IDF rather than count vectorization. It weights un

    vec = TfidfVectorizer()
    X = vec.fit_transform(train_data, train_labels)
    dev_X = vec.transform(dev_data)

    lr = LogisticRegression(C=100, solver='liblinear', multi_class='auto', penalty='l
    lr.fit(X, train_labels)

    pred = lr.predict(dev_X)

    f1 = metrics.f1_score(dev_labels, pred, average='weighted')
    print(f'The f1 score of the TF-IDF vectorizer is {f1}')
    print()
    print('The R ratio describes how "confidently incorrect" the classifier is. A hig
    print('label got a low score, and an incorrect label got a high score. It seems t
    print('in each newsgroup.')
    print()
    print('One method of improving this is to remove vocabular words that appear in *
    print('too weighty.')
```

```
    probs = lr.predict_proba(dev_X)

    # Helper function to calculate the R ratio
    def calculate_R(probs, label):
        return max(probs) / probs[label]

    # Sort the label probabilities by their R ratio
    sorted_probs = [(calculate_R(p, dev_labels[i]), i) for i, p in enumerate(probs)]
    sorted_probs.sort(key=lambda p: p[0])

    # Print the 3 documents with the highest R ratios
    for R, i in sorted_probs[::-1][:3]:
        print('=' * 40)
        print(f'R = {R}, label = {label_names[dev_labels[i]]}, predicted = {label_nam
        print('=' * 40)
        print(dev_data[i])
        print()
        print()
        print()

    ### STUDENT END ###

P7()
```

```
to make "verbatim" copies for personal use. People can recuperate the
actual costs of printing (paper, copy center charges), but may not charge
anything for their time in making copies, or in any way realize a profit
from the use of this book. See the permissions notice in the book itself
for the precise terms.

Negotiations are currently underway with a Mormon publisher vis-a-vis the
printing and distribution of bound books. (Sorry, I'm out of the wire-bound
"first editions.") I will make another announcement about the availability
of printed copies once everything has been worked out.

FTP information: connect via anonymous ftp to carnot.itc.cmu.edu, then "cd
pub" (you won't see anything at all until you do).

"The Easy-to-Read Book of Mormon" is currently available in postscript and
RTF (rich text format). (ASCII, LaTeX, and other versions can be made
available; contact dba@andrew.cmu.edu for details.) You should be able to
print the postscript file on any postscript printer (such as an Apple
Laserwriter); let dba know if you have any difficulties. (The postscript in
the last release had problems on some printers; this time it should work
better.) RTF is a standard document interchange format that can be read in
by a number of word processors, including Microsoft Word for both the
Macintosh and Windows. If you don't have a postscript printer, you may be
able to use the RTF file to print out a copy of the book.

-r--r--r--  1 dba                   1984742 Apr 27 13:12 etrbom.ps
-r--r--r--  1 dba                   1209071 Apr 27 13:13 etrbom.rtf

For more information about how this project came about, please refer to my
article in the current issue of _Sunstone_, entitled "Delighting in
Plainness: Issues Surrounding a Simple Modern English Book of Mormon."
```

```
Send all inquiries and comments to:

    Lynn Matthews Anderson
    5806 Hampton Street
    Pittsburgh, PA 15206



========================================
R = 325.0038462992751, label = talk.religion.misc, predicted = comp.graphics
========================================
Can anyone provide me a ftp site where I can obtain a online version
of the Book of Mormon. Please email the internet address if possible.



========================================
R = 287.3072077917014, label = alt.atheism, predicted = talk.religion.misc
========================================

The 24 children were, of course, killed by a lone gunman in a second story
window, who fired eight bullets in the space of two seconds...
```

ANSWER:

Part 8 EXTRA CREDIT:

Produce a Logistic Regression model to implement your suggestion from Part 7.

check 0s completed at 1:56 PM