Homework 3: Supervised machine learning

UIC CS 418, Spring 2022

According to the ***Academic Integrity Policy*** *of this course, all work submitted for grading must be done individually, unless otherwise specified. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at* [*https://dos.uic.edu/conductforstudents.shtml (https://dos.uic.edu/conductforstudents.shtml)*](https://dos.uic.edu/conductforstudents.shtml).

This homework is an individual assignment for all graduate students. Undergraduate students are allowed to work in pairs and submit one homework assignment per pair. There will be no extra credit given to undergraduate students who choose to work alone. The pairs of students who choose to work together and submit one homework assignment together still need to abide by the Academic Integrity Policy and not share or receive help from others (except each other).

Due Date

This assignment is due at 11:59pm Tuesday, March 29th, 2022.

What to Submit

You need to complete all code and answer all questions denoted by **`Q#`** (each one is under a bike image) in this notebook. When you are done, you should export **`hw3.ipynb`** with your answers as a PDF file, upload that file `hw3.pdf` to *Homework 3 - Written Part* on Gradescope, tagging each question.

You need to copy all functions that are part of questions Q1-Q9 to `hw3.py`. That includes `process()`, `process_all()`, `create_features()`, `create_labels()`, `class MajorityLabelClassifier()`, `learn_classifier()`, `evaluate_classifier()`, `best_model_selection()` and `classify_tweets()`. You need to upload a completed Jupyter notebook (`hw3.ipynb` file) and `hw3.py` to *Homework 3 - code* on Gradescope. To help you get started, we have provided a template file (`hw3_template.py`) containing imports, some hints, and function skeletons.

For undergraduate students who work in a team of two, only one student needs to submit the homework and just tag the other student on Gradescope.

Autograding

Questions will be graded based on both manual grading and an Autograder which will run on your `hw3.py` file. This assignment is graded on the basis of correctness and 80/100 points are given by the autograder. The remaining 20 points will be manually graded (10 points for Q8, 2 points for Q9 and 8 points for correctly running everything in the Jupyter notebook).

Most of the questions are graded independently. This means that if you have an error in a question, it will not be propagated to another question. However, the final question Q9 will check your overall pipeline and is rather expensive to run on Gradescope. Therefore, you should disable its auto-grading on Gradescope until you have implemented and passed Q1 to Q7. A function `test_pipeline()` is provided in the hw3_template.py file that returns `False` by default to disable auto-grading of Q9. Once you complete the implementation of Q1 to Q7, you can enable auto-grading of the whole pipeline by setting `test_pipeline()` to return `True`.”

The test cases will take a bit longer to execute. Make use of the resources wisely by first testing your functions in your notebook or making local test cases.

In [1]:
```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import sklearn
import string
import re # helps you filter urls
from IPython.display import display, Latex, Markdown
```

Classifying tweets [100%]

In this problem, you will be analyzing Twitter data extracted using [this (https://dev.twitter.com/overview/api)](https://dev.twitter.com/overview/api) api. The data contains tweets posted by the following six Twitter accounts: `realDonaldTrump, mike_pence, GOP, HillaryClinton, timkaine, TheDemocrats`

For every tweet, there are two pieces of information:

- `screen_name` : the Twitter handle of the user tweeting and
- `text` : the content of the tweet.

The tweets have been divided into two parts - train and test available to you in CSV files. For train, both the `screen_name` and `text` attributes were provided but for test, `screen_name` is hidden.

The overarching goal of the problem is to "predict" the political inclination (Republican/Democratic) of the Twitter user from one of his/her tweets. The ground truth (i.e., true class labels) is determined from the `screen_name` of the tweet as follows

- `realDonaldTrump, mike_pence, GOP` are Republicans
- `HillaryClinton, timkaine, TheDemocrats` are Democrats

Thus, this is a binary classification problem.

The problem proceeds in three stages:

- **Text processing (25%)**: We will clean up the raw tweet text using the various functions offered by the [nltk (http://www.nltk.org/genindex.html)](http://www.nltk.org/genindex.html) package.
- **Feature construction (25%)**: In this part, we will construct bag-of-words feature vectors and training labels from the processed text of tweets and the `screen_name` columns respectively.
- **Classification (50%)**: Using the features derived, we will use [sklearn (http://scikit-learn.org/stable/modules/classes.html)](http://scikit-learn.org/stable/modules/classes.html) package to learn a model which classifies the tweets as desired.

You will use two new python packages in this problem: `nltk` and `sklearn`, both of which should be available with anaconda. However, NLTK comes with many corpora, toy grammars, trained models, etc, which have to be downloaded manually. This assignment requires NLTK's stopwords list, POS tagger, and WordNetLemmatizer. Install them using:

In [2]:
```
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
# Verify that the following commands work for you, before moving on.

lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()
stopwords=nltk.corpus.stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/malika/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/malika/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/malika/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt to /Users/malika/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Let's begin!

A. Text Processing [25%]

You first task to fill in the following function which processes and tokenizes raw text. The generated list of tokens should meet the following specifications:

1. The tokens must all be in lower case.
2. The tokens should appear in the same order as in the raw text.
3. The tokens must be in their lemmatized form. If a word cannot be lemmatized (i.e, you get an exception), simply catch it and ignore it. These words will not appear in the token list.
4. The tokens must not contain any punctuations. Punctuations should be handled as follows: (a) Apostrophe of the form `'s` must be ignored. e.g., `She's` becomes `she`. (b) Other apostrophes should be omitted. e.g, `don't` becomes `dont`. (c) Words must be broken at the hyphen and other punctuations.
5. The tokens must not contain any part of a url.

Part of your work is to figure out a logical order to carry out the above operations. You may find `string.punctuation` useful, to get hold of all punctuation symbols. Look for [regular expressions (https://docs.python.org/3/library/re.html)](https://docs.python.org/3/library/re.html) capturing urls in the text. Your tokens must be of type `str`. Use `nltk.word_tokenize()` for tokenization once you have handled punctuation in the manner specified above.

You would want to take a look at the `lemmatize()` function [here (https://www.nltk.org/_modules/nltk/stem/wordnet.html)](https://www.nltk.org/_modules/nltk/stem/wordnet.html). In order for `lemmatize()` to give you the root form for any word, you have to provide the context in which you want to lemmatize through the `pos` parameter: `lemmatizer.lemmatize(word, pos=SOMEVALUE)`. The context should be the part of speech (POS) for that word. The good news is you do not have to manually write out the lexical categories for each word because [nltk.pos_tag() (https://www.nltk.org/book/ch05.html)](https://www.nltk.org/book/ch05.html) will do this for you. Now you just need to use the results from `pos_tag()` for the `pos` parameter. However, you can notice the POS tag returned from `pos_tag()` is in different format than the expected pos by `lemmatizer`.

> pos (Syntactic category): n for noun files, v for verb files, a for adjective files, r for adverb files.

You need to map these pos appropriately. `nltk.help.upenn_tagset()` provides description of each tag returned by `pos_tag()`.

Q1 (15%):

In [3]:
```
# Convert part of speech tag from nltk.pos_tag to word net compatible
# Simple mapping based on first letter of return tag to make grading c
# Everything else will be considered noun 'n'
posMapping = {
# "First_Letter by nltk.pos_tag":"POS_for_lemmatizer"
    "N":'n',
    "V":'v',
    "J":'a',
    "R":'r'
}

# 14% credits
def process(text, lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()):
    """ Normalizes case and handles punctuation
    Inputs:
        text: str: raw text
        lemmatizer: an instance of a class implementing the lemmatize(
                    (the default argument is of type nltk.stem.wordnet
    Outputs:
        list(str): tokenized text
    """

    cleared_text = text
```

```
    # Clear the string
    cleared_text = cleared_text.lower()
    cleared_text = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*
    cleared_text = cleared_text.replace("'s", "")
    cleared_text = cleared_text.replace("'", "")
    cleared_text = cleared_text.translate(str.maketrans(string.punctua

    tokens = nltk.word_tokenize(cleared_text)
    tags = nltk.pos_tag(tokens)

    for i in range(len(tokens)):
        if tags[i][1][0] in posMapping:
            tokens[i] = lemmatizer.lemmatize(tokens[i], posMapping[tag
        else:
            try:
                tokens[i] = lemmatizer.lemmatize(tokens[i], 'n')
            except:
                pass

    return(tokens)

print(process("'RT @SenSanders: My dad was born in Poland. Do you know
print("['rt', 'sensanders', 'my', 'dad', 'be', 'bear', 'in', 'poland',
```

```
['rt', 'sensanders', 'my', 'dad', 'be', 'bear', 'in', 'poland', 'do',
'you', 'know', 'how', 'many', 'people', 'ever', 'ask', 'me', 'whether
', 'or', 'not', 'i', 'be', 'bear', 'in', 'america', 'nobody', 'ever',
'aske…']
['rt', 'sensanders', 'my', 'dad', 'be', 'bear', 'in', 'poland', 'do',
'you', 'know', 'how', 'many', 'people', 'ever', 'ask', 'me', 'whether
', 'or', 'not', 'i', 'be', 'bear', 'in', 'america', 'nobody', 'ever',
'aske…']
```

You can test the above function as follows. Try to make your test strings as exhaustive as possible. Some checks are:

In [4]:
```
# 1% credit
print(process("I'm doing well! How about you?"))
# ['im', 'do', 'well', 'how', 'about', 'you']

print(process("Education is the ability to listen to almost anything w
# ['education', 'be', 'the', 'ability', 'to', 'listen', 'to', 'almost'

print(process("been had done languages cities mice"))
# ['be', 'have', 'do', 'language', 'city', 'mice']

print(process("It's hilarious. Check it out http://t.co/dummyurl"))
# ['it', 'hilarious', 'check', 'it', 'out']

print(process("See it Sunday morning at 8:30a on RTV6 and our RTV6 app
# ['see', 'it', 'sunday', 'morning', 'at', '8', '30a', 'on', 'rtv6', '
# Here '…' is a special unicode character not in string.punctuation an
```

```
['im', 'do', 'well', 'how', 'about', 'you']
['education', 'be', 'the', 'ability', 'to', 'listen', 'to', 'almost',
'anything', 'without', 'lose', 'your', 'temper', 'or', 'your', 'self'
, 'confidence']
['be', 'have', 'do', 'language', 'city', 'mice']
['it', 'hilarious', 'check', 'it', 'out']
['see', 'it', 'sunday', 'morning', 'at', '8', '30a', 'on', 'rtv6', 'a
nd', 'our', 'rtv6', 'app', 'http', '…']
```

Q2 (10%):

You will now use the `process()` function we implemented to convert the pandas dataframe we just loaded from tweets_train.csv file. Your function should be able to handle any data frame which contains a column called `text`. The data frame you return should replace every string in `text` with the result of `process()` and retain all other columns as such. Do not change the order of rows/columns. Before writing `process_all()`, load the data into a DataFrame and look at its format:

In [5]:
```
tweets = pd.read_csv("tweets_train.csv", na_filter=False)
display(tweets.head())
```

	screen_name	text
0	GOP	RT @GOPconvention: #Oregon votes today. That m...
1	TheDemocrats	RT @DWStweets: The choice for 2016 is clear: W...
2	HillaryClinton	Trump's calling for trillion dollar tax cuts f...
3	HillaryClinton	.@TimKaine's guiding principle: the belief tha...
4	timkaine	Glad the Senate could pass a #THUD / MilCon / ...

In [6]:
```
# 9% credits
def process_all(df, lemmatizer=nltk.stem.wordnet.WordNetLemmatizer()):
    """ process all text in the dataframe using process() function.
    Inputs
        df: pd.DataFrame: dataframe containing a column 'text' loaded
        lemmatizer: an instance of a class implementing the lemmatize(
                    (the default argument is of type nltk.stem.wordnet
    Outputs
        pd.DataFrame: dataframe in which the values of text column hav
                      the output from process() function. Other colu
    """
    copy_data = df.copy()
    copy_data['text'] = copy_data['text'].apply(process)

    return copy_data
```

In [7]:
```
# test your code
# 1% credit
processed_tweets = process_all(tweets)
print(processed_tweets.head())

#       screen_name                                                text
# 0             GOP  [rt, gopconvention, oregon, vote, today, that,...
# 1    TheDemocrats  [rt, dwstweets, the, choice, for, 2016, be, cl...
# 2  HillaryClinton  [trump, call, for, trillion, dollar, tax, cut,...
# 3  HillaryClinton  [timkaine, guide, principle, the, belief, that...
# 4        timkaine  [glad, the, senate, could, pass, a, thud, milc...
```

```
      screen_name                                                text
0             GOP  [rt, gopconvention, oregon, vote, today, that,...
1    TheDemocrats  [rt, dwstweets, the, choice, for, 2016, be, cl...
2  HillaryClinton  [trump, call, for, trillion, dollar, tax, cut,...
3  HillaryClinton  [timkaine, guide, principle, the, belief, that...
4        timkaine  [glad, the, senate, could, pass, a, thud, milc...
```

B. Feature Construction [25%]

The next step is to derive feature vectors from the tokenized tweets. In this section, you will be constructing a bag-of-words TF-IDF feature vector. But before that, as you may have guessed, the number of possible words is prohibitively large and not all of them may be useful for our classification task. We need to determine which words to retain, and which to omit. A common heuristic is to construct a frequency distribution of words in the corpus and prune out the head and tail of the distribution. The intuition of the above operation is as follows. Very common words (i.e. stopwords) add almost no information regarding similarity of two pieces of text. Similarly with very rare words. NLTK has a list of in-built stop words which is a good substitute for head of the distribution. We will consider a word rare if it occurs only in a single document (row) in whole of `tweets_train.csv` .

Q3 (15%):

Construct a sparse matrix of features for each tweet with the help of `sklearn.feature_extraction.text.TfidfVectorizer` (documentation [here (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)). You need to pass a parameter `min_df=2` to filter out the words occuring only in one document in the whole training set. Remember to ignore the stop words as well. You must leave other optional parameters (e.g., `vocab` , `norm` , etc) at their default values. But you may need to use parameters like `lowercase` and `tokenizer` to handle `processed_tweets` that is a `list` of tokens (not raw text).

In [8]:
```
# 14% credits
def create_features(processed_tweets, stop_words):
    """ creates the feature matrix using the processed tweet text
    Inputs:
        processed_tweets: pd.DataFrame: processed tweets read from tra
        stop_words: list(str): stop_words by nltk stopwords (after pro
    Outputs:
        sklearn.feature_extraction.text.TfidfVectorizer: the TfidfVect
            we need this to tranform test tweets in the same way as tr
        scipy.sparse.csr.csr_matrix: sparse bag-of-words TF-IDF featur
    """

    def id(text):
        return text

    vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(lower
    X = vectorizer.fit_transform(processed_tweets['text'])

    return vectorizer, X
```

In [9]:
```
# execute this code
# 1% credit
# It is recommended to process stopwords according to our data cleanin
processed_stopwords = set(np.concatenate([process(word) for word in st

(tfidf, X) = create_features(processed_tweets, processed_stopwords)
# Ignore warning
tfidf, X
# Output (should be similar):
# (TfidfVectorizer(lowercase=False, min_df=2,
#                  stop_words={'a', 'about', 'above', 'after', 'again'
#                              'ain', 'all', 'an', 'and', 'any', 'aren
#                              'at', 'be', 'because', 'before', 'below
#                              'both', 'but', 'by', 'can', 'couldn', '
#                              'd', 'didn', 'didnt', 'do', 'doesn', ..
#                  tokenizer=<function create_features.<locals>.<lambd
#  <17298x8114 sparse matrix of type '<class 'numpy.float64'>'
#   with 170355 stored elements in Compressed Sparse Row format>)
```

```
/opt/anaconda3/envs/cs418env/lib/python3.9/site-packages/sklearn/feat
ure_extraction/text.py:396: UserWarning: Your stop_words may be incon
sistent with your preprocessing. Tokenizing the stop words generated
tokens ['b', 'c', 'e', 'f', 'g', 'h', 'j', 'l', 'n', 'p', 'r', 'u', '
v', 'w'] not in stop_words.
  warnings.warn(
```

Out[9]:
```
(TfidfVectorizer(lowercase=False, min_df=2,
                 stop_words={'a', 'about', 'above', 'after', 'again',
'against',
                             'ain', 'all', 'an', 'and', 'any', 'aren'
, 'arent',
                             'at', 'be', 'because', 'before', 'below'
, 'between',
                             'both', 'but', 'by', 'can', 'couldn', 'c
ouldnt',
                             'd', 'didn', 'didnt', 'do', 'doesn', ...
},
                 tokenizer=<function create_features.<locals>.id at 0
x7f9d85b97af0>),
 <17298x8114 sparse matrix of type '<class 'numpy.float64'>'
        with 170355 stored elements in Compressed Sparse Row format>)
```

Q4 (10%):

Also for each tweet, assign a class label (0 or 1) using its `screen_name` . Use 0 for realDonaldTrump, mike_pence, GOP and 1 for the rest.

In [10]:
```
# 9% credits
def create_labels(processed_tweets):
    """ creates the class labels from screen_name
    Inputs:
        processed_tweets: pd.DataFrame: tweets read from train file, c
    Outputs:
        numpy.ndarray(int): dense binary numpy array of class labels
    """
    labels = processed_tweets[['screen_name']].to_numpy().ravel()
    for i in range(len(labels)):
        if (labels[i] == 'GOP') | (labels[i] == 'realDonaldTrump') | (
            labels[i] = 0
        else:
            labels[i] = 1
    labels = labels.astype(int)
    return labels
```

In [11]:
```
# execute this code
# 1% credit
y = create_labels(processed_tweets)
y
# 0        0
# 1        1
# 2        1
# 3        1
# 4        1
#         ..
# 17293    0
# 17294    0
# 17295    0
# 17296    1
# 17297    0
# Name: screen_name, Length: 17298, dtype: int32
```

Out[11]:
```
array([0, 1, 1, ..., 0, 1, 0])
```

C. Classification [50%]

And finally, we are ready to put things together and learn a model for the classification of tweets. The classifier you will be using is `sklearn.svm.SVC` [(http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) (Support Vector Machine).

At the heart of SVMs is the concept of kernel functions, which determines how the similarity/distance between two data points in computed. `sklearn` 's SVM provides four kernel functions: `linear` , `poly` , `rbf` , `sigmoid` (details [here (http://scikit-learn.org/stable/modules/svm.html#svm-kernels)](http://scikit-learn.org/stable/modules/svm.html#svm-kernels)) but you can also implement your own distance function and pass it as an argument to the classifier.

Through the various functions you implement in this part, you will be able to learn a classifier, score a classifier based on how well it performs, use it for prediction tasks and compare it to a baseline.

Specifically, you will carry out the following tasks (Q5-9) in order:

1. Implement and evaluate a simple baseline classifier MajorityLabelClassifier.
2. Implement the `learn_classifier()` function assuming `kernel` is always one of { `linear` , `poly` , `rbf` , `sigmoid` }.
3. Implement the `evaluate_classifier()` function which scores a classifier based on accuracy of a given dataset.
4. Implement `best_model_selection()` to perform cross-validation by calling `learn_classifier()` and `evaluate_classifier()` for different folds and determine which of the four kernels performs the best.
5. Go back to `learn_classifier()` and fill in the best kernel.

Q5 (10%):

To determine whether your classifier is performing well, you need to compare it to a baseline classifier. A baseline is generally a simple or trivial classifier and your classifier should beat the baseline in terms of a performance measure such as accuracy. Implement a classifier called `MajorityLabelClassifier` that always predicts the class equal to **mode** of the labels (i.e., the most frequent label) in training data. Part of the code is done for you. Implement the `fit` and `predict` methods. Initialize your classifier appropriately.

In [12]:
```
# Skeleton of MajorityLabelClassifier is consistent with other sklearn
# 8% credits

import statistics as st

class MajorityLabelClassifier():
    """
    A classifier that predicts the mode of training labels
    """
    def __init__(self):
        """
        Initialize your parameter here
        """
        self.m = None



    def fit(self, X, y):
        """
        Implement fit by taking training data X and their labels y and
        i.e. store your learned parameter
        """
        self.m = st.mode(y)


    def predict(self, X):
        """
        Implement to give the mode of training labels as a prediction
        return labels
        """
        return np.array([self.m]  * len(X))



# 2% credits
# Report the accuracy of your classifier by comparing the predicted la
baselineClf = MajorityLabelClassifier()
# Use fit and predict methods to get predictions and compare it with t

baselineClf.fit(processed_tweets['text'], y)
training_accuracy = sklearn.metrics.accuracy_score(y, baselineClf.pred


print(training_accuracy)
# should give 0.5001734304543878
```

```
0.5001734304543878
```

Q6 (10%):

Implement the `learn_classifier()` function assuming `kernel` is always one of { `linear` , `poly` , `rbf` , `sigmoid` }. Stick to default values for any other optional parameters.

In [13]:
```
# 9% credits
def learn_classifier(X_train, y_train, kernel):
    """ learns a classifier from the input features and labels using t
    Inputs:
        X_train: scipy.sparse.csr.csr_matrix: sparse matrix of feature
        y_train: numpy.ndarray(int): dense binary vector of class labe
        kernel: str: kernel function to be used with classifier. [line
    Outputs:
        sklearn.svm.SVC: classifier learnt from data
    """
    classif = sklearn.svm.SVC(kernel=kernel, verbose=False)
    classif.fit(X_train, y_train)
    return classif
```

In [14]:
```
# execute code
# 1% credit
classifier = learn_classifier(X, y, 'linear')
```

Q7 (10%):

Now that we know how to learn a classifier, the next step is to evaluate it, ie., characterize how good its classification performance is. This step is necessary to select the best model among a given set of models, or even tune hyperparameters for a given model.

There are two questions that should now come to your mind:

1. **What data to use?**
   - **Validation Data**: The data used to evaluate a classifier is called **validation data** (or hold-out data), and it is usually different from the data used for training. The model or hyperparameter with the best performance in the held out data is chosen. This approach is relatively fast and simple but vulnerable to biases found in validation set.
   - **Cross-validation**: This approach divides the dataset in k groups (so, called k-fold cross-validation). One of group is used as test set for evaluation and other groups as training set. The model or hyperparameter with the best average performance across all k folds is chosen. For this question you will perform 4-fold cross validation to determine the best kernel. We will keep all other hyperparameters default for now. This approach provides robustness toward biasness in validation set. However, it takes more time.
2. **And what metric?** There are several evaluation measures available in the literature (e.g., accuracy, precision, recall, F-1,etc) and different fields have different preferences for specific metrics due to different goals. We will go with accuracy. According to wiki, **accuracy** of a classifier measures the fraction of all data points that are correctly classified by it; it is the ratio of the number of correct classifications to the total number of (correct and incorrect) classifications. `sklearn.metrics` provides a number of performance metrics.

Now, implement the following function.

In [15]:
```
# 9% credits
def evaluate_classifier(classifier, X_validation, y_validation):
    """ evaluates a classifier based on a supplied validation data
    Inputs:
        classifier: sklearn.svm.classes.SVC: classifer to evaluate
        X_validation: scipy.sparse.csr.csr_matrix: sparse matrix of fe
        y_validation: numpy.ndarray(int): dense binary vector of class
    Outputs:
        double: accuracy of classifier on the validation data
    """
    return sklearn.metrics.accuracy_score(y_validation, classifier.pre
```

In [16]:
```
# test your code by evaluating the accuracy on the training data
# 1% credit
accuracy = evaluate_classifier(classifier, X, y)
print(accuracy)
# should give around 0.9545034107989363
```

```
0.9545034107989363
```

Q8 (10%):

Now it is time to decide which kernel works best by using the cross-validation technique. Write code to split the training data into 4-folds (75% training and 25% validation) by shuffling randomly. For each kernel, record the average accuracy for all folds and determine the best classifier. Since our dataset is balanced (both classes are in almost equal propertion), `sklearn.model_selection.KFold` [doc (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html) can be used for cross-validation.

In [17]:
```
kf = sklearn.model_selection.KFold(n_splits=4, random_state=1, shuffle
kf
```

Out[17]:
```
KFold(n_splits=4, random_state=1, shuffle=True)
```

Then use the following code to determine which classifier is the best.

In [18]:
```
# 10% credits
def best_model_selection(kf, X, y):
    """
    Select the kernel giving best results using k-fold cross-validatic
    Other parameters should be left default.
    Input:
    kf (sklearn.model_selection.KFold): kf object defined above
    X (scipy.sparse.csr.csr_matrix): training data
    y (array(int)): training labels
    Return:
    best_kernel (string)
    """
#     [YOUR CODE HERE]



    for kernel in ['linear', 'rbf', 'poly', 'sigmoid']:
#         [YOUR CODE HERE]
        # Use the documentation of KFold cross-validation to split ..
        # training data and test data from create_features() and creat
        # call learn_classifer() using training split of kth fold
        # evaluate on the test split of kth fold
        # record avg accuracies and determine best model (kernel)
    #return best kernel as string

#Test your code
best_kernel = best_model_selection(kf, X, y)
best_kernel
```

```
  File "/var/folders/qz/h154pclx6_j0cwkg7wyshknm0000gn/T/ipykernel_29
401/2764996877.py", line 27
    best_kernel = best_model_selection(kf, X, y)
                                                ^
IndentationError: expected an indented block
```

Q9 (10%)

We're almost done! It's time to write a nice little wrapper function that will use our model to classify unlabeled tweets from tweets_test.csv file.

In [19]:
```
# 8% credits
def classify_tweets(tfidf, classifier, unlabeled_tweets):
    """ predicts class labels for raw tweet text
    Inputs:
        tfidf: sklearn.feature_extraction.text.TfidfVectorizer: the Tf
        classifier: sklearn.svm.SVC: classifier learned
        unlabeled_tweets: pd.DataFrame: tweets read from tweets_test.c
    Outputs:
        numpy.ndarray(int): dense binary vector of class labels for un
    """
    return classifier.predict(tfidf.transform(process_all(unlabeled_tw
```

In [20]:
```
# Fill in best classifier in your function and re-trian your classifie
# Get predictions for unlabelled test data
# 2% credits
best_kernel = 'linear'
classifier = learn_classifier(X, y, best_kernel)
unlabeled_tweets = pd.read_csv("tweets_test.csv", na_filter=False)
y_pred = classify_tweets(tfidf, classifier, unlabeled_tweets)
```

Did your SVM classifier perform better than the baseline (while evaluating with training data)? Explain in 1-2 sentences how you reached this conclusion.

YOUR ANSWER HERE

In [21]:
```
"My SVM classifier performed better because its accuracy rate is equal
```

Out[21]:
```
'My SVM classifier performed better because its accuracy rate is equa
l to around 95% while Baseline classifier predicted with accuracy 50%
'
```

In [ ]:
```
```