ICE BREAKING FOR

DEVOIR 1 — IFT6285

DISCLAIMER

- The purpose of this course **is not** to familiarize you with machine learning, neither do I force you to use a classifier in this assignment, although it is likely the easiest thing to do.
- Those slides assume sklearn is installed in your (python) environment. This may be challenging on Windows (although I do believe it is feasible). On my computer it was the matter of running 2 command lines (*pip3 sklearn, and pip3 joblib*)

RUSHING ON YOUR FAVORITE CLASSIFIER IS CERTAINLY EASY, LIKELY FRUITFUL,

. . .

BUT MAKES YOU ONE OF US . . .

WC -L DATA/T*_POSTS.CSV

128158 data/test_posts.csv
512629 data/train_posts.csv

HEAD -N 1 DATA/TRAIN_POSTS.CSV

A BLOG CAN BE QUITE LONG

"long time.. i have been busy with work, school, rehersal, other random crap, and when i do have free time, my parents are on the computer. so shaddup! *memories* last night i started getting some memorys of my freind david, one of my favorites being the time my dog ate his bird. i look back and laugh, but it was pretty sad at the time. speaking of pets, i really want the tortoise at the pet store, how cool would that be, a pet tortoise?! and they live for hundreds of years, so it would make a great family heirloom, and sice they can do whatever to turtle shells so they don't rot (dry them out???) so when it died, that generation could still keep the shell. and they would have a reason to keep it, since it is a family herloom, it would be encrusted with jewels. horray! unfortunatly, the tortoise costs 140 bucks, not including food and so on. of course it would not need a coniner, sice it is a family heirloom, and i wil teach it to poop a certain place. horray. also, i never had a chance to use the self centered idea to find out who joah likes, so im just ganna come clean and tell him i like him. ",0

MAIN STATISTICS (FOR MORE SEE TP2)

CLASS DISTRIBUTION

	train	test
0	35.1	35.1
1	46.9	46.9
2	17.9	17.9

LENGTH DISTRIBUTION

	train	test
<=100	55.10	55.
<= 200	77.89	77.
<= 300	87.22	87.
<= 400	91.65	91.
avr.	163.0	162

CAT DATA/TRAIN_POSTS.CSV | DUMMY.PY

```
#!/usr/bin/env python3
# dummy.py

import sys
import pandas as pd
from sklearn.dummy import DummyClassifier
from joblib import dump

# 1) read stdin
df = pd.read_csv(sys.stdin, names=['blog', 'class'])

# 2) fit a dummy classifier
clf = DummyClassifier(strategy='most_frequent')
clf.fit(df[blog], df[classe])

# 3) dump it
dump(clf, './models/dummy-most.clf')
```

SKLEARN IS YOUR FRIEND

ONLY TAKES A FEW SECONDS TO RUN (TIME TO READ THE DATA)

CAT DATA/TEST_POSTS.CSV | TEST.PY MODELS/DUMMY-MOST.CLF

```
#!/usr/bin/env python3
# dummy.py

import sys, pickle, pandas as pd
from sklearn.dummy import DummyClassifier
from joblib import load

model_name = get_args() # to be written

# 1) read stdin
df = pd.read_csv(sys.stdin, names=['blog', 'class'])

# 2) fit a dummy classifier
clf = load(model_name)
y = clf.predict(df[blog])

# 3) output the predicition
base = os.path.split(model_name)[1]
out = f"{out_dir}/{base}.out"
pickle.dump([clf.classes_,y], open(out, 'wb'))
```

GENERATES OUT/DUMMY-MOST.OUT

CAT DATA/TEST_POSTS.CSV | EVAL.PY OUT/DUMMY-MOST.OUT

- You have to decide which metric(s) to use for evaluating your models
  - **accuracy** is the percentage of correct decisions over the total number of decisions to take (#of test examples)
    - If the dataset is **unbalanced**, it can artificially be quite high
  - **precision** is the % of correct decisions made over the total number of decisions taken (<= #test examples)
  - **recall** is the % of correct decisions made over the total number of decisions to take (#of test examples)
  - **f-measure** (F1) is often reported (harmonic mean of precision and recall)
  - When reporting precision and recall (therefore F1), you **need to clarify** which **task** is being asked to your classifier. So if for instance you consider important to find children, you could frame a task as « finding the authors that are children (class 0) », in which case precision and recall are defined as:
    - P = #children correctly identified / #identifications
    - R = #children correctly identified / #children in the test set

OUT/DUMMY-MOST.OUT: 67979 BAD EX OVER 128158 ONES. ACC: 46.96 %

THIS BEING UNDERSTOOD, USE SOMETHING LIKE:

```
from sklearn.metrics import classification_report
...
print(classification_report(truth, predicted))
```

```
CAT DATA/TEST_POSTS.CSV | EVAL.PY OUT/DUMMY-MOST.OUT
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/metrics/
classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. ‘precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00     45040
           1       0.47      1.00      0.64     60179
           2       0.00      0.00      0.00     22939

    accuracy                           0.47    128158
   macro avg       0.16      0.33      0.21    128158
weighted avg       0.22      0.47      0.30    128158
```

WARNS JUST BECAUSE OF THE DUMMY CLASSIFIER ALWAYS PREDICTS CLASS 1

VECTORIZER + LOGISTIC REGRESSION

- A **vectorizer** represents a text into a vector (of fixed size, defaulting to the number of different words in your collection). There are several of them built-in in **sklearn**, including:

  - CountVectorizer simply counts the words in a document v[i] is the count of the ith word in the document
  - TfidfVectorizer normalize the counts by taking into account the frequency of words in the collection (words seen in many documents will be discounted)

- LogisticRegression is one of the many classifiers available in **sklearn**. One of its advantage is that it allows to query the weights given to features, and when features are discreet (here words), it is rather instructive (see later)

CAT DATA/TRAIN_POSTS.CSV | TRAIN.PY . . .

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression

analyzer, min_df, … = get_args()    # to be implemented

# a number of options can control a vectorizer, I reckon you investigate them
vectorizer = TfidfVectorizer(analyzer=analyzer, min_df=min_df,
                                        max_df=max_df, max_features=max_features,
                                        ngram_range=(ngram_inf,ngram_sup),
                                        stop_words='english')

df = pd.read_csv(sys.stdin, names=['blog', 'class'])
X_train = vectorizer.fit_transform(df[blog])                    # isn't life beautiful ?

# several meta-parameters can influence the performance of Logit (investigate)
clf = LogisticRegression(C=5,class_weight='balanced', solver='newton-cg',
                                    multi_class='multinomial', n_jobs=-1, random_state=40, verbose=1)
clf.fit(X_train, df[classe])                                    # isn't life beautiful ?

# dump the vectorizer and the model (for use at test time)
dump(vectorizer, vec_name)
dump(clf, clf_name)
```

ALREADY QUITE SOME VARIANTS TO TEST…

- You could represent your text by their words, their chars, their bigram of words / chars, etc.
- Each representation can be parametrized
- The classifier has its own meta-parameters
- And we might want to play with other classifiers as well, eventually combining several …

I SUGGEST YOU START WITH THE BASICS (DEFAULT SETTINGS), AND THEN EXPLORE AS MUCH AS YOU CAN SYSTEMATICALLY (WITHOUT CODING SPECIFICALLY).

Notes:

- A one-file python notebook **is not** the best way of investigating many variants systematically.
- **T**his kind of meta-parameters investigation requires sane coding practices, and should typically be conducted on a **validation set** and **is** definitely **not green-AI**

LET'S TRAIN A FEW VARIANTS...

cat data/train_posts.csv | train.py –analyzer=word

10 MIN. TO TRAIN

34M model-straight-tfidf-analyzerword-ngram1-1-min_df0.0-max_df1.0-max_featuresNone-512629.vec

12M model-straight-tfidf-analyzerword-ngram1-1-min_df0.0-max_df1.0-max_featuresNone-512629.clf

cat data/train_posts.csv | train.py –analyzer=word –ngram_sup=2

84 MIN. TO TRAIN
+10 MIN. TO SAVE

1,1G model-straight-tfidf-analyzerword-ngram1-2-min_df0.0-max_df1.0-max_featuresNone-512629.vec

374M model-straight-tfidf-analyzerword-ngram1-2-min_df0.0-max_df1.0-max_featuresNone-512629.clf

LET'S TRAIN A FEW VARIANTS...

15 MIN. TO TRAIN

cat data/train_posts.csv | train.py –analyzer=char –ngram_sup=3

6,8M model-straight-tfidf-analyzerchar-ngram1-3-min_df0.0-max_df1.0-max_featuresNone-512629.vec

2,7M model-straight-tfidf-analyzerchar-ngram1-3-min_df0.0-max_df1.0-max_featuresNone-512629.clf

cat data/train_posts.csv | train.py --ngram_sup=2 --maxfeat=50000

RUNNING A MODEL

```
#1) parse the command line, prepare filenames to read/save
model_name, ... = get_args()       # to be implemented

#2) load the vectorizer and the classifier
vectorizer = load(f'{model_name}.vec')
clf = load(f'{model_name}.clf')

#3) read test data
df = pd.read_csv(sys.stdin, names=['blog', 'class'])

 X_test = vectorizer.transform(df[blog]) # apply the vectorizer
 y_test = clf.predict(X_test)                    # run the classifier

#4) save the predictions
base = os.path.split(model_name)[1]
out = f"{out_dir}/{base}.out"
pickle.dump([clf.classes_,y_test], open(out, 'wb'))
```

ex: cat data/test_posts.csv | test.py models/model-straight-tfidf-analyzerword-ngram1-1-min_df0.0-max_df1.0-max_featuresNone-512629

EVALUATING A PREDICTION

ex: cat data/test_posts.csv | eval.py out/model-straight-tfidf-analyzerword-ngram1-1-min_df0.0-max_df1.0-max_featuresNone-512629.out

	Config	Acc.
Word	1g	68.55
	1g - 50k feat	66.16
	1,2g	**71.92**
	1,2g-50k feat	65.50
Char	1-3g	63.21

THE BEST I HAVE (NO EXPLORATION)

	precision	recall	f1-score	support
0	0.80	0.79	0.79	45040
1	0.73	0.73	0.73	60179
2	0.54	0.55	0.55	22939
micro avg	0.72	0.72	0.72	128158
macro avg	0.69	0.69	0.69	128158
weighted avg	0.72	0.72	0.72	128158

```
[[35441  7602  1997]
 [ 7292 44005  8882]
 [ 1593  8617 12729]]
```

ACCESS TO THE KIND OF FEATURES FOUND USEFUL

class 0 : haha, anyways, homework, school, maths, xd, ap, marten, nicki, linds, thats, prom, exun, anywho, chem, current song, arv, karan, wad, bye, gunna, mrs, yay, im, awesome, w00t, hahaz, haiz, noe, gamespot, lolz, xanga, rosie, lol, current music, lindsey, ur, yea, jumper991, babysitting, soo, info link, alright, theres, jackie, laura, kuronue, xbubzx, jonah, idk, heinz, josh, random, ashley, camp, little kids, math, affectionately anna, seniors, tk, chemistry, colleges, tmk, abby, den, pointless, g2g, hahaha, ppl, gonna, theo, ttyl, boring, hmmn

class 1 : apartment, law school, mungo, daf, vlad, ching, semester, baity, posted paul, andrei, india, roommates, dena, parents house, internship, abt, holla, killy, fonz, bf, como, office, jax, pia, cyodfs, wedding, adolph, fro, urllink mail, undergrad, new wave, thesis, grad school, jonnie, roommate, pictures baby, shayne, cara, perth, damo, midterm, dori, darth, drinks, clubbing, high school, coz, furze, melbourne, lab, jerel, weekend, rini, boz, kicha, icq, coffee count, shawna, currently listening, triumph good, vegas, work, thing necessary, nhl, deane, jb, evil triumph, bless america, bangalore, campus, hostel, today msn, prof, professors

class 2 : duf, diva, corsair, guam, hubby, ok mm, hal, 2004 non, djs, non girlfriend, shep, daughter, venerable, copyright 2004, evermean, eric digest, dog news, dear lei, rick, son, scw, husband, tuesday quote, katelyn, katya, giulio, _____, heff, digest, pandyland, jayel, inda, shai, hax, liza, keisha, kids, email spanners, spanners, link courtesy, maiko, years, jennie, allot, brook, jp, lei, gethtmlforicon, folks, sharky, meds, submissive, leslie, drunkenfish, workout, mean mamma, acentos, greg, vicente, ecw, treadmill, mary torres, thanks urllink, dayton, katy, levengals, total far, linktocomments urllink,

WORD2VEC-INFUSED CLASSIFIER

- Trained Word2Vec (thanks to gensim) on the 512k blogs available
  - d=300, window=2, neg=200, min_count=10
  - A bit of normalisation done in the Vectorizer
- Fed the representation into a logistic regressor

	precision	recall	f1-score	support
0	0.53	0.58	0.55	45040
1	0.54	0.28	0.37	60179
2	0.27	0.58	0.37	22939
accuracy			0.44	128158
macro avg	0.45	0.48	0.43	128158
weighted avg	0.49	0.44	0.43	128158

```
#1) train and save a model
cat train_spacy.csv |
        toVec.py --gensim models/genw2v-size300-window2-neg200-mincount100.w2v |
        train-from-w2v.py Data_nobackup/models/genw2v-size300-window2-neg200-mincount100.clf

#2) load the model and apply it to the test
cat test_spacy.csv |
        toVec.py --gensim models/genw2v-size300-window2-neg200-mincount100.w2v |
        test-from-w2v.py Data_nobackup/models/genw2v-size300-window2-neg200-mincount100.clf

#3) evaluate the produced output
cat test_spacy.csv | eval.py  out/genw2v-size300-window2-neg200-mincount100.clf.out
```

TOVEC.PY

```
#1) read the embeddings
if trained_with_gensim:
    w2v_model = Word2Vec.load(model_name)
else:
    w2v_model = KeyedVectors.load_word2vec_format(model_name, binary=True)

#2) read the dataset
df = pd.read_csv(sys.stdin, names=[blog,classe])

#3) transform it
mean_vec_tr = MeanEmbeddingVectorizer(w2v_model)
doc_vec = mean_vec_tr.transform(df[blog])

#4 ) output in a csv-like format the vector and the class
arr = np.c_[ doc_vec, df[classe]]
np.savetxt(sys.stdout.buffer, arr, delimiter=',')
```

MEANEMBEDDINGVECTORIZER ADAPTED FROM HTTPS://GITHUB.COM/TOMLIN/PLAYGROUND/BLOB/MASTER/04-MODEL-COMPARISON-WORD2VEC-DOC2VEC-TFIDFWEIGHTED.IPYNB

TRAIN-FROM-W2V.PY

```
#1) read the matrix from stdin
arr = np.loadtxt(sys.stdin.buffer, delimiter=',')

x = arr[::,:-1]  # tout sauf la derniere colonne
y = arr[::,-1].astype('int')   # juste la derniere colonne (label, en int)

#2) train a model
clf = LogisticRegression(C=5.0,class_weight='balanced', solver='newton-cg',
                         multi_class='multinomial', n_jobs=-1, random_state=40, verbose=1)
clf.fit(x, y)

# 3) save it
dump(clf, model_name)
```

TEST-FROM-W2V.PY

```
# 1) read the w2v vectors and the class
arr = np.loadtxt(sys.stdin.buffer, delimiter=',')
x = arr[::,:-1]  # tout sauf la derniere colonne
y = arr[::,-1].astype('int')   # juste la derniere colonne (label, en int)

# 2) load the classifier
clf = load(model_name)

#3) run the prediction
y_test = clf.predict(x)

#4) output a result
base = os.path.split(model_name)[1]
out = f"{out_dir}/{base}.out"
pickle.dump([clf.classes_, y_test], open(out, 'wb'))
```

WORD2VEC-INFUSED CLASSIFIER (TAKE 2)

- Also considered the pre-trained embeddings distributed by Google
  - Google news - 3M words
- Performance slightly worse

	precision	recall	f1-score	support
0	0.50	0.60	0.55	45040
1	0.54	0.24	0.33	60179
2	0.26	0.56	0.36	22939
accuracy			0.42	128158
macro avg	0.44	0.46	0.41	128158
weighted avg	0.48	0.42	0.41	128158

WELL...

- Investigating preprocessing
- Investigating metapameters
- Testing other feature-based models
    - XGBoost is currently very popular
        - Consider as well SVM, random Forest, etc.
- Deep Learning
    - Stacking a softmax on top of an RNN encoder should give you a boost (expect +2% accuracy), and if not, just pretend the data is too small.
    - Buy a GPU first, or use a platform like Colab.
- Using rules (why not?)
- Analyze the failure of your best strategy

HOW DOES IT FEEL ?

TESTS: 34 GROUPS SUBMITTED A REPORT

- Some did not submitted their results to the *mystere* test set
  - Emmanuel, Charles
- Some did submit a test file, but with bad number of lines
  - Marie-Ève (8149)
  - Zachary (110000)
  - Elyes (10000)
- Format was (somehow) underspecified
  - Some reported text without csv encoding as the first column
  - With or without header (my fault)
- Did accommodate all this
  - removing headers manually, safely removing first column
  - some potential issues (but I do not think so)

ACCURACY

Elyes-Lamouchi	5482	32.80	FFN
Philippe-Lelièvre	5184	36.46	BERT
David-Ferland	4778	41.43	LR
Marie-Eve-Malette-Campeau	3724	54.35	LR
Fanny-Salvail-Bérard	3375	58.63	XLNet
Daniel-Galarreta-Piquette	3219	60.54	NB?
Soheila-Kiani	3133	61.60	CNN
Francis-de-Ladurantaye	3069	62.38	BERT
Michel-Ma	2978	63.50	SVM or LR
Zachary-Barillaro	2956	63.77	NB
Gustavo-Alonso-Patino-Ramirez	2940	63.96	Doc2vec + ?
Luis-Dos-Santos	2922	64.18	LR
Kodjine-Dare	2677	67.19	???
Martin-Weyssow	2614	67.96	???
Hubert-Corriveau	2528	69.01	???
Khalil-Slimi	2491	69.47	LR
Lucas-Pages	2391	70.69	kenLM
Amini-Mohammad	2327	71.48	???
Olivier-Salaün	2269	72.19	???
Yan-Zeng	2261	72.28	???
Philippe-Gagné Chafouleas-Genevieve	2235	72.60	SVM?
Jean-Pierre-Thach	2233	72.63	LR
Aboubaker-Aden-Houssein	2222	72.76	DL (unclear)
Lu-Yuchen	2123	73.98	fastText
Tianjian-Gao	2016	75.29	SVC
Adrien-Mainka	1991	75.59	biGRU?
Nithin-Anchuri_	1958	76.00	SVM
Leila-Camille-Hanis-Fabing	1936	76.27	FastText?
Yutao-Zhu	1916	76.51	BERT
Khalil	1914	76.54	tf-idf++
Simon-Pelletier	1899	76.72	BERT++

PRECISION, RECALL AND F-MEASURE ON CLASS-2 (>=30)

	P	R	FM	
Olivier-Salaün	0.48	0.31	0.38	???
Aboubaker-Aden-Houssein	0.44	0.37	0.40	DL (unclear)
Michel-Ma	0.29	0.62	0.40	SVM or LR
Lucas-Pages	0.35	0.50	0.41	kenLM
Amini-Mohammad	0.39	0.47	0.43	???
Hubert-Corriveau	0.34	0.59	0.43	???
Tianjian-Gao	0.50	0.37	0.43	SVC
Leila-Camille-Hanis-Fabing	0.52	0.38	0.44	fastText
Philippe-Gagné	0.47	0.41	0.44	SVM?
Khalil-Slimi	0.35	0.65	0.45	LR
Nithin-Anchuri	0.50	0.41	0.45	SVM
Yan-Zeng	0.43	0.47	0.45	???
Adrien-Mainka	0.54	0.42	**0.47**	**biGRU?**
Khalil	0.50	0.45	**0.47**	**LR-tfidf++**
Simon-Pelletier	0.57	0.41	**0.47**	**BERT++**
Yutao-Zhu	0.57	0.40	**0.47**	**BERT**

SENSITIVITY TO SHORT TEXTS ?

	blog<100 1285	blog<500 3934	
Yutao-Zhu	62.57	71.86	BERT
Olivier-Salaün	62.88	68.02	???
Lu-Yuchen	63.04	69.52	fastText
Khalil	63.50	72.17	LR++
Aboubaker-Aden-Houssein	63.66	69.19	DL(unclear)
Adrien-Mainka	64.05	70.74	biGRU?
Simon-Pelletier	64.51	71.84	BERT++
Tianjian-Gao	65.29	71.58	SVC
Leila-Camille-Hanis-Fabing	65.76	**72.32**	**fastText**
Nithin-Anchuri	**66.07**	71.91	SVM

COOL VIZUALISATION (MERCI MARTIN)

Figure 2: Wordclouds de certains clusters pour k=20

- K-means on Doc2vec
- Some cleaning would have been better
- Would be interesting to see clusters per class

LACKING TIME, BUT SO FAR …

- Quite impressed by what was done by some groups
- Not analyzing the best solution is disappointing
- Normalisation does not seem to help
- LR is a good system to start (or even end) with
  - Some variation among groups on this
- BERT seems the best solution
  - What about BERT-large ?
  - Not necessarily easy to make it work