ICE BREAKING FOR
DEVOIR 1 — IFT6285
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 . . .
"long time.. i have been busy with work, school, rehearsal, 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 memories 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 since 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, since 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

<table>
<thead>
<tr>
<th></th>
<th>train</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35.1</td>
<td>35.1</td>
</tr>
<tr>
<td>1</td>
<td>46.9</td>
<td>46.9</td>
</tr>
<tr>
<td>2</td>
<td>17.9</td>
<td>17.9</td>
</tr>
</tbody>
</table>

#### Length Distribution

<table>
<thead>
<tr>
<th></th>
<th>train</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=100</td>
<td>55.10</td>
<td>55.</td>
</tr>
<tr>
<td>&lt;=200</td>
<td>77.89</td>
<td>77.</td>
</tr>
<tr>
<td>&lt;=300</td>
<td>87.22</td>
<td>87.</td>
</tr>
<tr>
<td>&lt;=400</td>
<td>91.65</td>
<td>91.</td>
</tr>
<tr>
<td>avr.</td>
<td>163.0</td>
<td>162</td>
</tr>
</tbody>
</table>
FIRST STEP: DUMMY BASELINE

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['class'])

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

ONLY TAKES A FEW SECONDS TO RUN (TIME TO READ THE DATA)
FIRST STEP: DUMMY BASELINE

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 prediction
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
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:

- \[ P = \frac{\text{#children correctly identified}}{\text{#identifications}} \]
- \[ R = \frac{\text{#children correctly identified}}{\text{#children in the test set}} \]
This being understood, use something like:

```python
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)

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>45040</td>
</tr>
<tr>
<td>1</td>
<td>0.47</td>
<td>1.00</td>
<td>0.64</td>
<td>60179</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>22939</td>
</tr>
</tbody>
</table>

|             | 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
A REAL BASELINE

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 $i$th 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)
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['class'])  # isn’t life beautiful ?

# dump the vectorizer and the model (for use at test time)
dump(vectorizer, vec_name)
dump(clf, clf_name)
A REAL BASELINE: VECTORIZER + LR

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.
- This kind of meta-parameters investigation requires sane coding practices, and should typically be conducted on a validation set and is definitely not green-AI.
A REAL BASELINE: VECTORIZER + LR

LET'S TRAIN A FEW VARIANTS...

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

**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**

**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
A REAL BASELINE: VECTORIZER + LR

LET’S TRAIN A FEW VARIANTS...

```bash
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

15 MIN. TO TRAIN
```

cat data/train_posts.csv | train.py --ngram_sup=2 --maxfeat=50000
A REAL BASELINE: VECTORIZER + LR

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

<table>
<thead>
<tr>
<th>Config</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1g</td>
<td>68.55</td>
</tr>
<tr>
<td>1g - 50k feat</td>
<td>66.16</td>
</tr>
<tr>
<td>1,2g</td>
<td>71.92</td>
</tr>
<tr>
<td>1,2g-50k feat</td>
<td>65.50</td>
</tr>
<tr>
<td>1-3g</td>
<td>63.21</td>
</tr>
</tbody>
</table>
SO FAR

THE BEST I HAVE (NO EXPLORATION)

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.80</td>
<td>0.79</td>
<td>0.79</td>
<td>45040</td>
</tr>
<tr>
<td>1</td>
<td>0.73</td>
<td>0.73</td>
<td>0.73</td>
<td>60179</td>
</tr>
<tr>
<td>2</td>
<td>0.54</td>
<td>0.55</td>
<td>0.55</td>
<td>22939</td>
</tr>
</tbody>
</table>

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]]
WHY I LOVE LR (OR DECISIONS TREES/STUMPS, ETC.)

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, xububzx, 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, local, room, passes laws, today miles, iphoneshape, tini, derville, wife, therapist henry newman, ke, love,
ANOTHER BASELINE

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

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.53</td>
<td>0.58</td>
<td>0.55</td>
<td>45040</td>
</tr>
<tr>
<td>1</td>
<td>0.54</td>
<td>0.28</td>
<td>0.37</td>
<td>60179</td>
</tr>
<tr>
<td>2</td>
<td>0.27</td>
<td>0.58</td>
<td>0.37</td>
<td>22939</td>
</tr>
</tbody>
</table>

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

```bash
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

```bash
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

```bash
cat test_spacy.csv | eval.py out/genw2v-size300-window2-neg200-mincount100.clf.out
```
#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=','
#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)
# 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'))
ANOTHER BASELINE

WORD2VEC-INFUSED CLASSIFIER (TAKE 2)

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

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.50</td>
<td>0.60</td>
<td>0.55</td>
<td>45040</td>
</tr>
<tr>
<td>1</td>
<td>0.54</td>
<td>0.24</td>
<td>0.33</td>
<td>60179</td>
</tr>
<tr>
<td>2</td>
<td>0.26</td>
<td>0.56</td>
<td>0.36</td>
<td>22939</td>
</tr>
</tbody>
</table>

accuracy         0.42  128158
macro avg        0.44  0.46  0.41  128158
weighted avg     0.48  0.42  0.41  128158
WHAT ELSE CAN BE DONE?

WELL...

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

**ACCURACY**
## Precision, Recall and F-Measure on Class-2 (>=30)

<table>
<thead>
<tr>
<th>Name</th>
<th>P</th>
<th>R</th>
<th>FM</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olivier-Salaün</td>
<td>0.48</td>
<td>0.31</td>
<td>0.38</td>
<td>???</td>
</tr>
<tr>
<td>Aboubaker-Aden-Houssein</td>
<td>0.44</td>
<td>0.37</td>
<td>0.40</td>
<td>DL (unclear)</td>
</tr>
<tr>
<td>Michel-Ma</td>
<td>0.29</td>
<td>0.62</td>
<td>0.40</td>
<td>SVM or LR</td>
</tr>
<tr>
<td>Lucas-Pages</td>
<td>0.35</td>
<td>0.50</td>
<td>0.41</td>
<td>kenLM</td>
</tr>
<tr>
<td>Amini-Mohammad</td>
<td>0.39</td>
<td>0.47</td>
<td>0.43</td>
<td>???</td>
</tr>
<tr>
<td>Hubert-Corriveau</td>
<td>0.34</td>
<td>0.59</td>
<td>0.43</td>
<td>???</td>
</tr>
<tr>
<td>Tianjian-Gao</td>
<td>0.50</td>
<td>0.37</td>
<td>0.43</td>
<td>SVC</td>
</tr>
<tr>
<td>Leila-Camille-Hanis-Fabing</td>
<td>0.52</td>
<td>0.38</td>
<td>0.44</td>
<td>fastText</td>
</tr>
<tr>
<td>Philippe-Gagné</td>
<td>0.47</td>
<td>0.41</td>
<td>0.44</td>
<td>SVM?</td>
</tr>
<tr>
<td>Khalil-Slimi</td>
<td>0.35</td>
<td>0.65</td>
<td>0.45</td>
<td>LR</td>
</tr>
<tr>
<td>Nithin-Anchuri</td>
<td>0.50</td>
<td>0.41</td>
<td>0.45</td>
<td>SVM</td>
</tr>
<tr>
<td>Yan-Zeng</td>
<td>0.43</td>
<td>0.47</td>
<td>0.45</td>
<td>???</td>
</tr>
<tr>
<td>Adrien-Mainka</td>
<td>0.54</td>
<td>0.42</td>
<td>0.47</td>
<td>biGRU?</td>
</tr>
<tr>
<td>Khalil</td>
<td>0.50</td>
<td>0.45</td>
<td>0.47</td>
<td>LR-tfidf++</td>
</tr>
<tr>
<td>Simon-Pelletier</td>
<td>0.57</td>
<td>0.41</td>
<td>0.47</td>
<td>BERT++</td>
</tr>
<tr>
<td>Yutao-Zhu</td>
<td>0.57</td>
<td>0.40</td>
<td>0.47</td>
<td>BERT</td>
</tr>
<tr>
<td>Name</td>
<td>blog&lt;100</td>
<td>blog&lt;500</td>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Yutao-Zhu</td>
<td>62.57</td>
<td>71.86</td>
<td>BERT</td>
<td></td>
</tr>
<tr>
<td>Olivier-Salaün</td>
<td>62.88</td>
<td>68.02</td>
<td>???</td>
<td></td>
</tr>
<tr>
<td>Lu-Yuchen</td>
<td>63.04</td>
<td>69.52</td>
<td>fastText</td>
<td></td>
</tr>
<tr>
<td>Khalil</td>
<td>63.50</td>
<td>72.17</td>
<td>LR++</td>
<td></td>
</tr>
<tr>
<td>Aboubaker-Aden-Houssein</td>
<td>63.66</td>
<td>69.19</td>
<td>DL(unclear)</td>
<td></td>
</tr>
<tr>
<td>Adrien-Mainka</td>
<td>64.05</td>
<td>70.74</td>
<td>biGRU?</td>
<td></td>
</tr>
<tr>
<td>Simon-Pelletier</td>
<td>64.51</td>
<td>71.84</td>
<td>BERT++</td>
<td></td>
</tr>
<tr>
<td>Tianjian-Gao</td>
<td>65.29</td>
<td>71.58</td>
<td>SVC</td>
<td></td>
</tr>
<tr>
<td>Leila-Camille-Hanis-Fabing</td>
<td>65.76</td>
<td>72.32</td>
<td>fastText</td>
<td></td>
</tr>
<tr>
<td>Nithin-Anchuri</td>
<td>66.07</td>
<td>71.91</td>
<td>SVM</td>
<td></td>
</tr>
</tbody>
</table>
- K-means on Doc2vec
- Some cleaning would have been better
- Would be interesting to see clusters per class
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