NLP For Economists

Lecture 5: NLP when we don't have labeled datasets

Sowmya Vajjala

MGSE - LMU Munich
Guest Course, October 2022

12th October 2022

Outline

- Why this topic?
- Data labeling and Augmentation
- Various approaches to address a "no/less data scenario"
- Weak supervision: A closer look
- A walk through of different methods for a single usecase

Why this topic?

- The starting point of any modern NLP system is data. But we don't always have readymade data.
- We have looked at how to collect data from various sources in Lecture 3.
- I have briefly mentioned automatic labeling of data too.
- We will delve deeper into this topic today.

Typical "no data to start with" scenarios

- a common NLP problem, but for a specific domain (e.g., financial sentiment analysis)
- building a common NLP system, but for a new language (e.g., a named entity recognizer for, say, French)
- building a common NLP system, for a new, low resource language (e.g., machine translation for English to Ojibwe language)
- A custom problem (e.g., classification into some focused categories.

etc

Building a labeled dataset - the traditional way

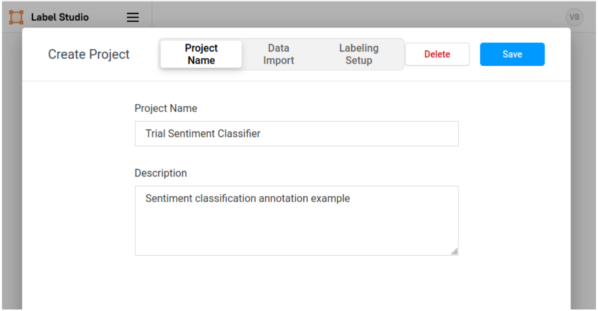
- Collect from existing sources (we've discussed this)
- Collect on your own from scratch.

Labeling Tools

- There are many such data annotation tools and companies.
- this listing gives an overview of pricing, pros and cons of some such tools.
- I used a tool called Label Studio in the past to understand how these tools work, and felt it is good to create a small dataset by ourselves.

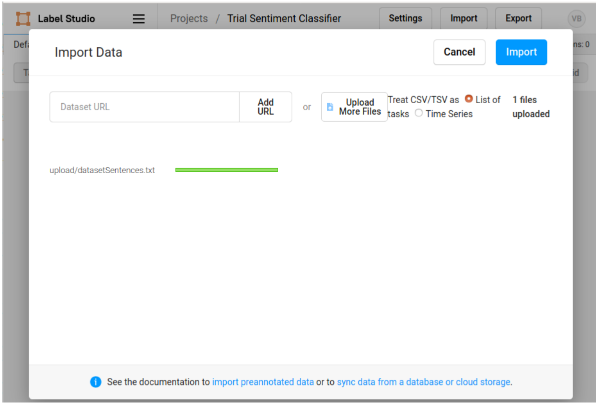
Annotating our own data: Label studio

Let us say I want to create data for sentiment classification task.

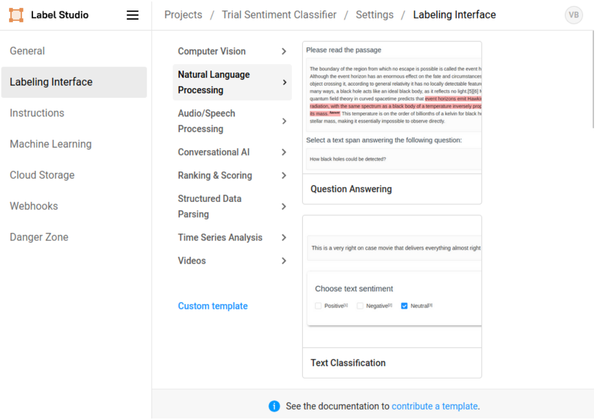


Importing the data

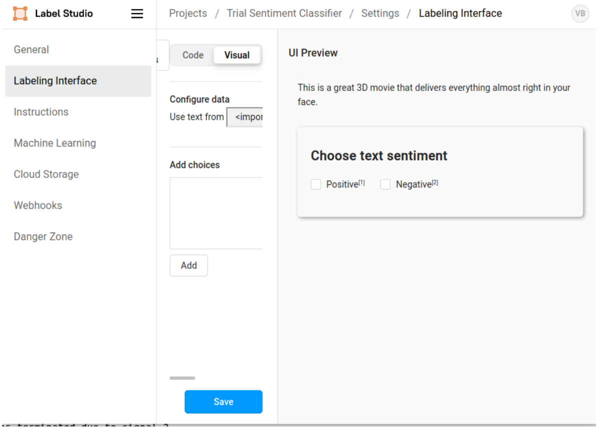
I just have a bunch of sentences, which I am importing into Label studio.



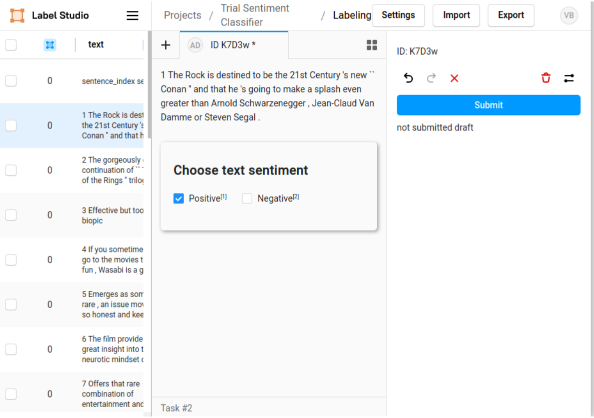
Choosing the task



Setting up stuff



Label!


Label Studio
Projects / Trial Sentiment Classifier / Labeling
Settings
Import
Export
VB
text
0 sentence_index se
0 1 The Rock is dest the 21st Century 's Conan " and that h
0 2 The gorgeously continuation of `` of the Rings " trilo
0 3 Effective but too biopic
0 4 If you sometime go to the movies t fun , Wasabi is a g
0 5 Emerges as som rare , an issue mo so honest and kee
0 6 The film provide great insight into t neurotic mindset
0 7 Offers that rare combination of entertainment and
AD ID K7D3w *
1 The Rock is destined to be the 21st Century 's new `` Conan " and that he 's going to make a splash even greater than Arnold Schwarzenegger , Jean-Claud Van Damme or Steven Segal .
Choose text sentiment
Positive[1]
Negative[2]
Task #2
ID: K7D3w
Submit
not submitted draft

Input and Output

sentence_index	sentence
1	The Rock is destined to be the 21st Century 's new `` Conan '' and
2	The gorgeously elaborate continuation of `` The Lord of the Rings '
3	Effective but too-tepid biopic
4	If you sometimes like to go to the movies to have fun , Wasabi is a
5	Emerges as something rare , an issue movie that 's so honest and ke
6	The film provides some great insight into the neurotic mindset of a
7	Offers that rare combination of entertainment and education .
8	Perhaps no picture ever made has more literally showed that the roa
9	Steers turns in a snappy screenplay that curls at the edges ; it 's
10	But he somehow pulls it off .
11	Take Care of My Cat offers a refreshingly different slice of Asian
12	This is a film well worth seeing , talking and singing heads and al
13	What really surprises about Wisegirls is its low-key quality and ge
14	-LRB- Wendigo is -RRB- why we go to the cinema : to be fed through
15	One of the greatest family-oriented , fantasy-adventure movies ever
16	Ultimately , it ponders the reasons we need stories so much .
17	An utterly compelling ` who wrote it ' in which the reputation of t
18	Illuminating if overly talky documentary .

text	id	sentiment	annotator	annotatio	created_a	updated_	lead_time
7Offers that rare combination of entertain	8	Positive	vbsowmy	7	2021-08-1	2021-08-1	299.235
8Perhaps no picture ever made has more l	9	Negative	vbsowmy	8	2021-08-1	2021-08-1	3.048
6The film provides some great insight into	7	Negative	vbsowmy	6	2021-08-1	2021-08-1	2.468
5Emerges as something rare , an issue mov	6	Negative	vbsowmy	5	2021-08-1	2021-08-1	11.045
4If you sometimes like to go to the movies	5	Positive	vbsowmy	4	2021-08-1	2021-08-1	5.703
3Effective but too-tepid biopic	4	Negative	vbsowmy	3	2021-08-1	2021-08-1	2.217
2The gorgeously elaborate continuation of	3	Positive	vbsowmy	2	2021-08-1	2021-08-1	6.609
1The Rock is destined to be the 21st Centu	2	Positive	vbsowmy	1	2021-08-1	2021-08-1	40.257

Label Studio: Summary

- There can be more annotators, and you can do other stuff like checking for agreement among human annotators etc
- This is just one popular tool, and I used its free version. There are other useful features to build our own data.

Is the data we created sufficient?

- depends on the task, language etc.
- sometimes, 1000 labeled examples is a lot.
- sometimes, 50K labeled examples is not that much.
- for tasks like language modeling (read: GPT-3 and so on), no amount of data is "a lot", it looks like.

Data Augmentation

- We may still not be able to get enough labeled data as manual labeling can be intensive and time consuming.
- Data augmentation is all about generating new data by slightly modifying existing (labeled) data.
- How?: Replacing words with synonyms, back translation etc.

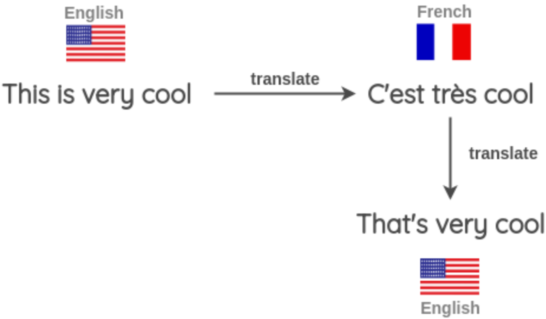
Data Augmentation: Synonym replacement

Textual Data Augmentation Example

	Sentence
Original	The quick brown fox jumps over the lazy dog
Synonym (PPDB)	The quick brown fox climbs over the lazy dog
Word Embeddings (word2vec)	The easy brown fox jumps over the lazy dog
Contextual Word Embeddings (BERT)	Little quick brown fox jumps over the lazy dog
PPDB + word2vec + BERT	Little easy brown fox climbs over the lazy dog

source

Data Augmentation: Back translation



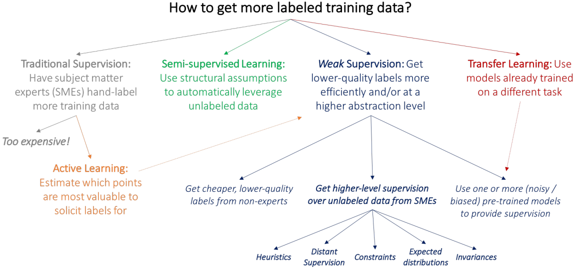
source

Data Augmentation:Summary

- It was shown to be useful for some NLP tasks
- It is also used in real-world application scenarios
- This repository presents a survey of data augmentation methods for NLP.
- Caveat: It doesn't always work for all tasks. For sentiment, machine translation etc, it may work. For something else, you may find it hard to generate such data.
- Transformations are usually minor alterations to input text.

Modeling with small(-er) datasets

Even data augmentation may not give you enough data, sometimes.



source

Weak Supervision: An Introduction

- Generally, most 'learning' methods used in NLP are data hungry. However, it is time consuming and also expensive to hand label so much of data for each new problem.

Weak Supervision: An Introduction

- Generally, most 'learning' methods used in NLP are data hungry. However, it is time consuming and also expensive to hand label so much of data for each new problem.
- Sometimes, we may have to update existing labels to suit changed guidelines or just update the dataset etc. (not so uncommon in real world). How do we handle the costs/time taken?

Weak Supervision: An Introduction

- Generally, most 'learning' methods used in NLP are data hungry. However, it is time consuming and also expensive to hand label so much of data for each new problem.
- Sometimes, we may have to update existing labels to suit changed guidelines or just update the dataset etc. (not so uncommon in real world). How do we handle the costs/time taken?
- "Weak supervision" refers to a machine learning approach which relies on "imprecise" training data, which is potentially "generated" automatically.

Weak Supervision: An Introduction

- Generally, most 'learning' methods used in NLP are data hungry. However, it is time consuming and also expensive to hand label so much of data for each new problem.
- Sometimes, we may have to update existing labels to suit changed guidelines or just update the dataset etc. (not so uncommon in real world). How do we handle the costs/time taken?
- "Weak supervision" refers to a machine learning approach which relies on "imprecise" training data, which is potentially "generated" automatically.
- An approach: write code based on observed patterns in data to label subsets of unlabeled data.
- ... and then use this code to create labeled training data for our ML model

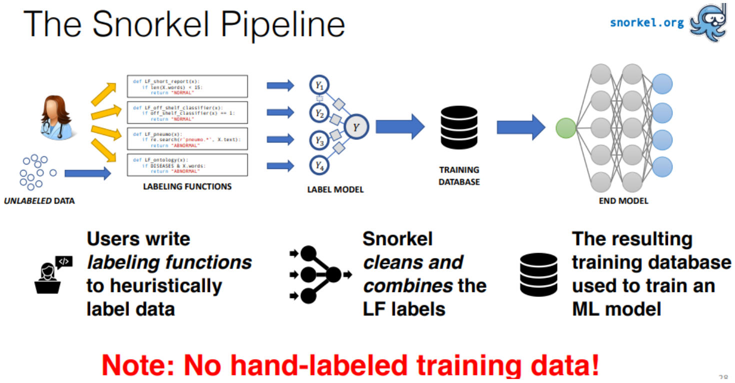
Weak Supervision: Continued

- Clearly, this is a noisy dataset. There may be labeling errors. What now?

Weak Supervision: Continued

- Clearly, this is a noisy dataset. There may be labeling errors. What now?
- Snorkel's approach:
  - Create a noisy training set through "labeling functions" (I showed one in the last class.)
  - Learn a model of this noise (to understand which of these functions are good in terms of labeling)
  - Uses this model to train a more powerful model which learns from the noise.

The next few slides will rely on This talk slides by Alex Ratner, one of the people behind Snorkel.

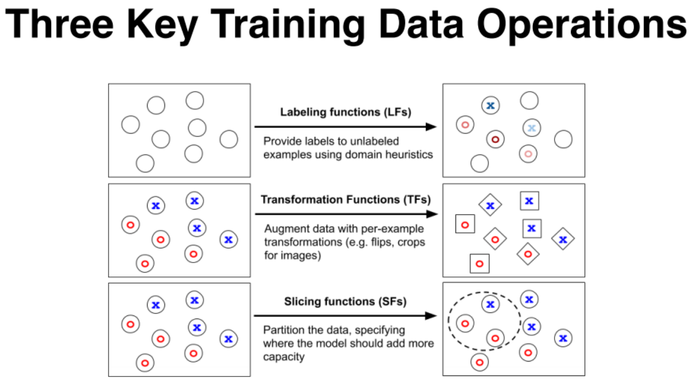


https://db.cs.washington.edu/events/workshop/2019/slides/alex-ratner.pdf

When is this useful?

- No training data
- Expensive training data (which needs specific expertise)
- Private data (which can't be exposed to crowd workers, for example)
- Constantly changing data

How to "create" data?



How to "create" data?- Labeling Functions

SuperGLUE Labeling Function (LF)

```
def lf_matching_trigrams(x):
    if trigram(x.sentences[0].target) == trigram(x.sentences[1].target):
        return TRUE
    else:
        return ABSTAIN
```

id: x1
Sentence 0: Can I **invite** you for dinner on Sunday night?
Sentence 1: The organizers **invite** submissions of papers.
Label: FALSE

`lf_matching_trigrams(x1) == ABSTAIN`

id: x2
Sentence 0: He felt a **stream** of air .
Sentence 1: The hose ejected a **stream** of water .
Label: TRUE

`lf_matching_trigrams(x2) == TRUE`

How to "create" data? - augmentation

SuperGLUE Transformation Function (TF)

```
def tf_days_of_the_week(x):
    yield x
    for DAY in DAYS_OF_WEEK:
        yield replace_with_synonym(x, word=DAY, synonyms=DAYS_OF_WEEK)
```

id: x1
Sentence 1: Can I **invite** you for dinner on Sunday night?
Sentence 2: The organizers **invite** submissions of papers.

`tf_days_of_the_week(x1)`

Sentence 1: Can I **invite** you for dinner on Sunday night?
Sentence 1: Can I **invite** you for dinner on Monday night?
Sentence 1: Can I **invite** you for dinner on Tuesday night?
Sentence 1: Can I **invite** you for dinner on Wednesday night?
Sentence 1: Can I **invite** you for dinner on Thursday night?
Sentence 1: Can I **invite** you for dinner on Friday night?
Sentence 1: Can I **invite** you for dinner on Saturday night?

How to "create" data? - slicing

SuperGLUE Slicing Function (SF)

```
def sf_target_is_noun(x):
    if x.sentences[0].target.pos == NOUN and x.sentences[1].target.pos == NOUN
        return NOUN_SLICE
    else:
        return ABSTAIN
```

id: x1
Sentence 0: Can I **invite** you for dinner on Sunday night?
Sentence 1: The organizers **invite** submissions of papers.

`sf_target_is_noun(x1) == ABSTAIN`

id: x2
Sentence 0: He felt a **stream** of air .
Sentence 1: The hose ejected a **stream** of water .

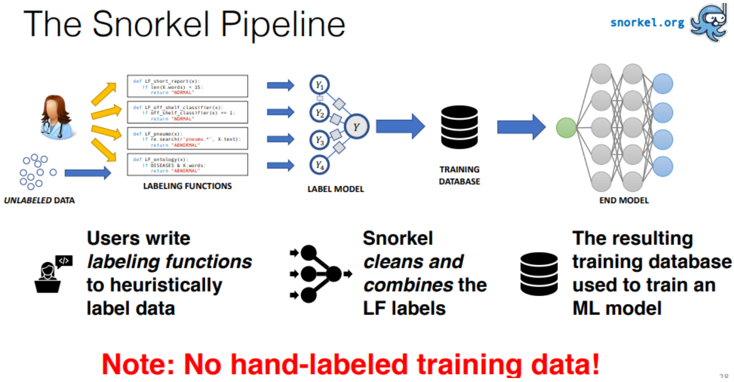
`sf_target_is_noun(x2) == NOUN_SLICE`

What is "slicing"?

- In real-world systems, some predictions/categories may be more important than others.
- However, when we build these learning models, we look at overall performance.
- So, "slicing" functions in Snorkel identify these subsets of data that we should particularly care about.
- Note: This is done after training data is ready, otherwise we cannot create these subsets!

an interesting tidbit: slice based learning was deployed in production systems at Apple in 2019.

Let us revisit the snorkel approach



What is happening at "Label model"?

There is actually no "labeled" data. What is this "label model" learning? and how?

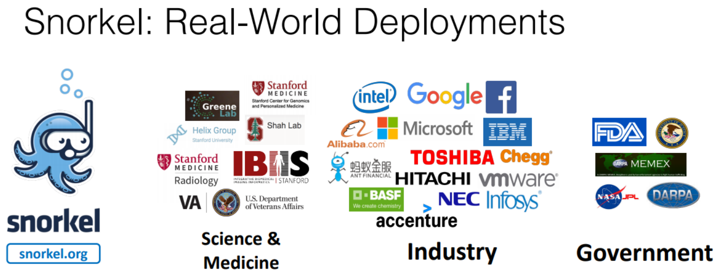
What is happening at "Label model"?

There is actually no "labeled" data. What is this "label model" learning? and how?

Key idea: learn from the agreements and disagreements among label functions about a single data point!

It all sounds good, does this really work?

Snorkel in Real world (2019)



https://db.cs.washington.edu/events/workshop/2019/slides/alex-ratner.pdf

Specific Examples - Industry Usecases

- Serving >1B queries (multiple languages) with weak supervision and data slicing systems at Apple: Overton: A Data System for Monitoring and Improving Machine-Learned Products
- Conversational agents at IBM: Bootstrapping Conversational Agents With Weak Supervision (AAAI 2019)
- Web content & event classification at Google: Snorkel DryBell: A Case Study in Deploying Weak Supervision at Industrial Scale (SIGMOD Industry 2019), and Google AI blog post
- Business intelligence at Intel: Osprey: Non-Programmer Weak Supervision of Imbalanced Extraction Problems (SIGMOD DEEM 2019)

Specific Examples - Clinical NLP

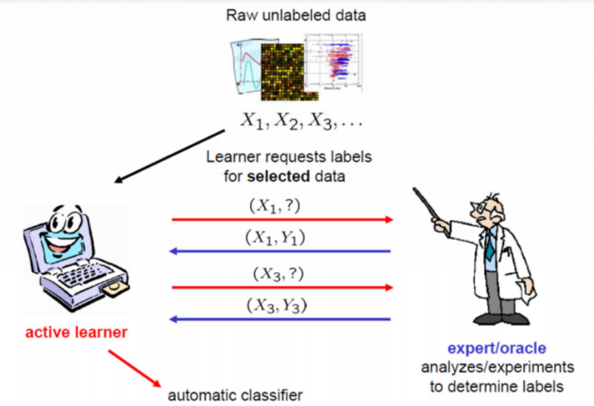
- Medical image triaging at Stanford Radiology: Cross-Modal Data Programming Enables Rapid Medical Machine Learning (Preprint)
- GWAS KBC with Stanford Genomics: A machine-compiled database of genome-wide association studies (Nature Communications 2019)
- Clinical text classification: A clinical text classification paradigm using weak supervision and deep representation (BMC MIDM 2019)
- SwellShark: A Generative Model for Biomedical Named Entity Recognition without Labeled Data SwellShark: A Generative Model for Biomedical Named Entity Recognition without Labeled Data

Snorkel Approach: A summary

- We may frequently see situations without training data.
- It is possible to generate training data through heuristics (string matching, regex etc), and a few "tricks" (like augmentation)
- Not all training instances are equally important. So, we can partition the data (slicing) and identify critical subsets.
- This is a two stage "modeling" - one for consolidating all labeling functions to build a training set, one for learning from this training set.

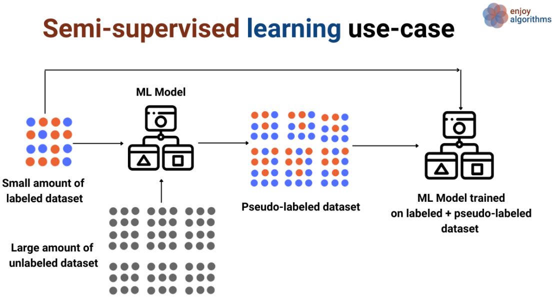
Other Approaches

Other methods: Active Learning



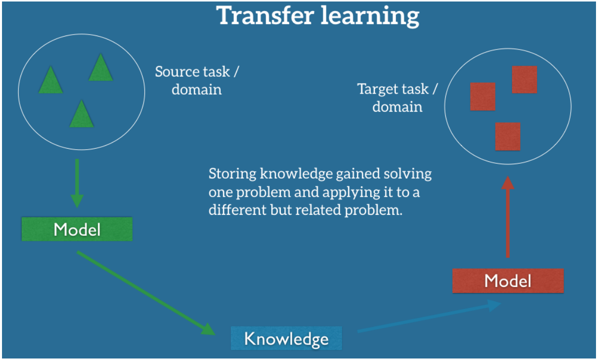
source: Professor Tom Mitchell's course slides

Other methods: Semi supervised Learning



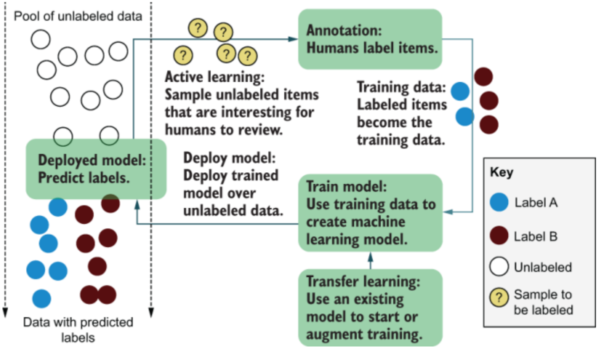
source

Other methods: Transfer Learning



source Note: Transfer can also be cross-lingual.

A Scenario Combining them all



source: Chapter 1 in "Human-in-the-Loop Machine Learning " by Robert Munro.

NLP For Economists

Sowmya Vajjala

Practical Advice-1

Let us say you have no data to start with. What is the way forward?

- Understand your requirements, and create a small, high-quality, manually inspected, labeled dataset (e.g., using label studio like tools)
- Evaluate an off-the shelf solution if it exists (e.g., a cloud service provider)
- Create automatically labeled data and build a model using weak supervision, evaluating with your high quality test data.

Practical Advice-2

- You managed to get some labeled data through automatic labeling or other means.
- You also managed a baseline weakly supervised model.
- Then, what?

Practical Advice-2

- You managed to get some labeled data through automatic labeling or other means.
- You also managed a baseline weakly supervised model.
- Then, what?
- Evaluate transfer learning if a similar model is available
- Consider if Semi-supervised learning and/or Active learning will be useful

Practical Advice-3

- Slowly, you built up a large collection of labeled or pseudo-labeled data.
- You can then explore more sophisticated ML/DL models

Before all this, think if you really need all this, too. Rule based matching may just be sufficient for your scenario! (Check spacy's rule based matching support!)

Questions?

A Case Study

The Problem

- Problem Sentiment classification of sentences into positive and negative
- Nature of the dataset: Labeled. I will use it as if a part of it were unlabeled, in some of the examples that follow.
- why did I choose such a common problem?
  1. it is common, so we will find some ready made solutions to compare with automatic labeling
  2. it is common, but the dataset I chose makes it slightly difficult to use off the shelf solutions.

The Dataset

- Sentiment Labelled Sentences Dataset.
- sentences with one of the two labels: 1 (positive), 0 (negative)
- The sentences come from three websites: amazon, imdb, yelp.
- For each website, there are 500 sentences per category.
- I will use the amazon part (500+500 - 1000 labeled examples) as my test data everywhere.
  (reminder: in real world, you may have to create such a dataset using tools like label studio/doccano etc, or if you are lucky, you already have internal labeled data)

Modeling Methods

1. No labeled data scenario (with just labeled test data)
   1.1 using a cloud service provider's sentiment analyzer
   1.2 using an off the shelf Python library (free)
   1.3 using weak supervision (unlabeled train + labeled test data)
2. Comparing with labeled data scenario (labeled train + labeled test)
   2.1 train your sentiment classifier from scratch
   2.2 transfer learning: use an existing pre-trained language model and **tune** it using your training data

Methods: Text representation

(for weak supervision model, and when we build other classifiers with labeled data)

1. bag of words
2. sentence transformers (sbert.net)

Why? - to illustrate one simple text representation, one state of the art neural text representation.

No data NLP: using off the shelf solutions

Sentiment Analysis with Azure Text Analytics

- - "Sentiment Analysis in version 3.x applies sentiment labels to text, which are returned at a sentence and document level, with a confidence score for each."
- labels: positive, negative, mixed, neutral
- "Confidence scores range from 1 to 0. Scores closer to 1 indicate a higher confidence in the label's classification, while lower scores indicate lower confidence."

source: Azure Text Analytics website

Sentiment Analysis with Azure Text Analytics

Things to think about:

- We need a model with only positive/negative labels. What should we do about mixed/neutral?
- Is this a permanent solution, or should we start thinking about collecting labeled data eventually?

Sentiment Analysis with Azure Text Analytics


```
key = "XXXXX"
endpoint = "XXXXX"
```

```
from azure.ai.textanalytics import TextAnalyticsClient
from azure.core.credentials import AzureKeyCredential
```

```
def authenticate_client():
    ta_credential = AzureKeyCredential(key)
    text_analytics_client = TextAnalyticsClient(
            endpoint=endpoint, credential=ta_credential)
    return text_analytics_client

client = authenticate_client()
```

```
testfilepath = "test_labelled.txt" #tab seperated file.
sentences = []
sentiments = [] #0 is negative, 1 is positive
preds = [] #0 neg, 1 positive, 2 neutral or mixed
preds_dict = {'positive':1, 'negative':0, 'neutral':2, 'mixed':2}
count = 0
for line in open(testfilepath):
    sentence, sentiment = line.strip().split("\t")
    pred = preds_dict[client.analyze_sentiment(documents = [sentence])[0].sentiment]
    preds.append(pred)
    sentences.append(sentence)
    sentiments.append(int(sentiment))
```

Sentiment Analysis with Azure Text Analytics

```
print(classification_report(sentiments, preds))
```

```
              precision    recall  f1-score   support

           0       0.96      0.81      0.88       500
           1       0.93      0.88      0.90       500
           2       0.00      0.00      0.00         0

    accuracy                           0.84      1000
   macro avg       0.63      0.56      0.59      1000
weighted avg       0.95      0.84      0.89      1000
```

```
import collections
print(collections.Counter(preds))
```

```
Counter({1: 473, 0: 420, 2: 107})
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(sentiments,preds,labels=[0,1,2]))
```

```
[[405  34  61]
 [ 15 439  46]
 [  0   0   0]]
```

(About 100 instances from my test set are labeled either "netural" or "mixed" as per Azure.)

Sentiment Analysis with TextBlob


```
pred = TextBlob(sentence).sentiment.polarity
if pred > 0:
    pred =1 #positive sentiment
elif pred < 0:
    pred= 0 #negative sentiment
else:
    pred = 2 #when polarity is 0 i.e, neutral
```

```
              precision    recall  f1-score   support

           0       0.93      0.41      0.57       500
           1       0.78      0.83      0.80       500
           2       0.00      0.00      0.00         0

    accuracy                           0.62      1000
   macro avg       0.57      0.41      0.46      1000
weighted avg       0.85      0.62      0.69      1000

[[204 121 175]
 [ 16 417  67]
 [  0   0   0]]
```

No Data NLP: Summary

We get a good start: over 80% accuracy with Azure

Pros: You don't have to worry about setting stuff up, building and maintaining the sentiment analyzer etc. Cons:

- This only works if you have problem that exactly meets the specifications of such an available tool
- No possibility of customization/modification, we don't know what it is doing.
- Depending on how much you use, costs may escalate

Weak Supervision, with Snorkel

Note: I am using "unlabeled" training data to programmatically create labels.
How?:

1. Write a few labeling functions based on heuristics (I wrote using existing lists of positive/negative words)
2. learn a label model, from the output of such labeling functions (Snorkel's learner)
3. convert learnt label distribution into training data ready to be used by any ML/DL approach.

Labeling Functions with Snorkel


```python
#a simple labeling function checking if a sentence has positive words
@labeling_function()
def postive(x):
    poswords = 0
    temp = x.text.lower().split()
    for word in temp:
        if word in positives:
            poswords +=1
    if poswords > 0:
        return POS
    else:
        return ABSTAIN

#a simple labeling function checking if a sentence has negative words
@labeling_function()
def negative(x):
    negwords = 0
    temp = x.text.lower().split()
    for word in temp:
        if word in negatives:
            negwords +=1
    if negwords > 0:
        return NEG
    else:
        return ABSTAIN
```

(where the poswords/negwords came from a standard list)

How good are the labeling functions?


```
from snorkel.labeling import LFAnalysis

lfs = [postive,negative, vaderlex]

LFAnalysis(L=L_train, lfs=lfs).lf_summary()
```

	j	Polarity	Coverage	Overlaps	Conflicts
postive	0	[1]	0.4695	0.4395	0.1055
negative	1	[0]	0.2725	0.2360	0.1165
vaderlex	2	[0, 1]	0.6320	0.5725	0.1190

How good are the labeling functions?

- Coverage: The fraction of the dataset the LF labels
- Overlaps: The fraction of the dataset where this LF and at least one other LF label overlap
- Conflicts: The fraction of the dataset where this LF and at least one other LF label disagree

(Clearly, these are not sufficient/good enough LFs, but I am still going ahead, as I am using this only as an illustration!)

How many labeling functions?

as many as you can, such that:

- no two functions overlap too much
- together they should achieve maximum coverage of the unlabeled examples.

Learning a label model

Step 1: Convert your (unlabeled) train and (labeled) test into the feature representation based on these LFs.

```
applier = PandasLFApplier(lfs=lfs)
  L_train = applier.apply(df=df_train)
  L_test = applier.apply(df=df_test)
```

Learning a label model

Step 2: Our goal is now to convert the labels from our LFs into a single noise-aware probabilistic (or confidence-weighted) label per data point.

- An easy way: majority vote on a per-data point basis: if more LFs voted POS than NEG for a data point, label it POS (and vice versa)
- Snorkel also has a more advanced label model, to learn such confidence weighted label represnetations, though.

Learning a label model

```
from snorkel.labeling.model import MajorityLabelVoter
from snorkel.labeling.model import LabelModel

majority_model = MajorityLabelVoter()
preds_train = majority_model.predict(L=L_train)

label_model = LabelModel(cardinality=2, verbose=True)
label_model.fit(L_train=L_train, n_epochs=500, log_freq=100, seed=123)


Y_test = df_test.label.values

majority_acc = majority_model.score(L=L_test, Y=Y_test, tie_break_policy="random")["accuracy"]
print(f"{'Majority Vote Accuracy:':<25} {majority_acc * 100:.1f}%")

label_model_acc = label_model.score(L=L_test, Y=Y_test, tie_break_policy="random")["accuracy"]
print(f"{'Label Model Accuracy:':<25} {label_model_acc * 100:.1f}%")
```

```
Majority Vote Accuracy:   72.1%
Label Model Accuracy:     71.7%
```

Since my LFs are not that good (and too few?), we don't see much difference between MajorityLabel or LabelModel, with the former being slightly better.
So, why can't we just use this as the final labeling model??

From label model to training data

- If we use this approach, the data points the model will "ABSTAIN" from labeling some data points. What do we do with them?
- Instead, we will use the outputs of the LabelModel as training labels to train a classifier which can generalize beyond the labeling function outputs.

From label model to training data

Step 1: Filter out the unlabeled data points:

```
 from snorkel.labeling import filter_unlabeled_dataframe

df_train_filtered, probs_train_filtered
      = filter_unlabeled_dataframe(X=df_train, y=probs_train, L=L_train)

from snorkel.utils import probs_to_preds
preds_train_filtered = probs_to_preds(probs=probs_train_filtered)
```

Sentiment classification with this "generated" training data

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

Y_test = df_test.label.values

for classifier in [LogisticRegression(C=1e3, solver="liblinear"), LinearSVC()]:
    classifier.fit(X=X_train, y=preds_train_filtered)
    print("Performance for ", type(classifier).__name__),
    print(f"Test Accuracy: {classifier.score(X=X_test, y=test_labels) * 100:.1f}%")
    #print(preds_train_filtered)
    #print(test_labels)
```

```
Performance for  LogisticRegression
Test Accuracy: 61.2%
Performance for  LinearSVC
Test Accuracy: 62.0%
```

	precision	recall	f1-score	support
0	0.93	0.49	0.64	500
1	0.65	0.96	0.78	500
accuracy			0.73	1000
macro avg	0.79	0.73	0.71	1000
weighted avg	0.79	0.73	0.71	1000

```
[[247 253]
 [ 20 480]]
```

(We managed to get to 73% without an actual labeled training dataset!)

pros and cons

Pros:

- Quick way to generate (labeled) training data programmatically.
- Tried and tested, used in many industry usecases for NLP.
- Useful to build a first solution quickly. (73% is not a bad start without data!)

Cons:

- We have to develop the labeled functions, and it won't be easy unless we have clear knowledge about the problem.
- This may not work very well for all kinds of NLP problems.

What if I just have my labeled training set?

How do these approaches compare to a more optimistic scenario where I actually have some labeled training data??

We can do two things in this case:

- train our own classifier with our training data
- fine-tune a large language model using our training data

(and test with the same test set as before!)

Training your own classifier -1 (with bag of words)

```
#BOW feature extraction
vect = CountVectorizer(preprocessor=clean, max_features=1000) # instantiate a vectoriezer
train_vector = vect.fit_transform(train_texts)# use it to extract features from training data
# transform test data (using training data's features)
test_vector = vect.transform(test_texts)

#Use a logistic regression classifier to train and test the model
logreg = LogisticRegression() # instantiate a logistic regression model
logreg.fit(train_vector, train_labels) # fit the model with training data
# Make predictions on test data
predicted = logreg.predict(test_vector)

#Print results.
print(classification_report(test_labels, predicted))
print(confusion_matrix(test_labels,predicted))
```

```
              precision    recall  f1-score   support

           0       0.71      0.82      0.76       500
           1       0.78      0.67      0.72       500

    accuracy                           0.74      1000
   macro avg       0.75      0.74      0.74      1000
weighted avg       0.75      0.74      0.74      1000

[[408  92]
 [166 334]]
```

Training your own classifier -1 (with sbert)

```
from sentence_transformers import SentenceTransformer

#Choose from the models here: https://www.sbert.net/docs/pretrained_models.html
model = SentenceTransformer('paraphrase-TinyBERT-L6-v2')

#Read the train/test files and extract labels and
#textual features using the sentence transformers model.
def read_data(filepath):
    data = []
    labels = []
    for line in open(filepath):
        sentence, label = line.strip().split("\t")
        labels.append(label)
        data.append(model.encode(sentence)) #feature extraction
    return data, labels
```

```
#feature extraction
train_data, train_labels = read_data("../files/train_labelled.txt")
test_data, test_labels = read_data("../files/test_labelled.txt")


#modeling
clf = LogisticRegression(random_state=0).fit(train_data, train_labels)

#prediction/evaluation
predicted = clf.predict(test_data)
print(classification_report(test_labels, predicted))
print(confusion_matrix(test_labels,predicted))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.87       500
           1       0.86      0.89      0.88       500

    accuracy                           0.88      1000
   macro avg       0.88      0.88      0.87      1000
weighted avg       0.88      0.88      0.87      1000

[[428  72]
 [ 53 447]]
```

Transfer learning with our dataset

- intuition: When I used sbert features earlier, I just used the representations a large language model learnt (using some large data set, on some tasks) "as is".
- The goal of fine-tuning is to take this large language model as its base, and "re-train" it to suit our classification task, using our training data.
- The pre-trained model's weights are then altered ("fine-tuned") while training for the task
- while all this may sound complex, there are easy to use implementations of transfer learning for many NLP tasks.

Transfer learning with our dataset

```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=3,              # total number of training epochs
    per_device_train_batch_size=16,  # batch size per device during training
    per_device_eval_batch_size=64,   # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    logging_steps=10,
)

model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

trainer = Trainer(
    model=model,                         # the instantiated □ Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=val_dataset,             # evaluation dataset
    compute_metrics=compute_metrics
)

trainer.train()
predictions = trainer.predict(test_dataset)
preds = np.argmax(predictions.predictions, axis=-1)
metric =load_metric('accuracy', 'f1')
print(metric.compute(predictions=preds, references=predictions.label_ids))
```

This gave me 92.7% accuracy on the test set!

Pros and Cons of training our own models

- Pros:
  - We have control over the modeling process.
  - We may get better performance, as this is a custom made model for us.
- Cons:
  - We need large enough labeled training sets as a starting point.
  - We also need more knowledge and expertise, and know what to experiment and what to discard.

Summary of all the approaches

When we have don't have labeled training data (but have labeled test set)

Approach	Accuracy
predictions from Azure	84%
predictions from TextBlob	69%
Weak supervision with Snorkel (with sentence transformers)	73%

When we have some amount of labeled training data (and with the same test set)

Approach	Accuracy
Training our own model (with bag of words features)	74%
Training our own model (with sentence transformers)	88%
Transfer learning	**92.7%**

(Note: We can use data augmentation in scenarios from both tables, I leave it as an exercise!)

Caveats on the approach I took

- I only want to show a range of methods to apply when you encounter a ”no labeled data” scenario.
- so i took a relatively easy example
- this is by no means a statement that azure works or transfer learning works.
- with careful heuristics, even weak supervision may give you much better performance than what you saw in this example!

Revisiting: Practical Advice-1

Let us say you have no data to start with. What is the way forward?

- Understand your requirements, and create a small, high-quality, manually inspected, labeled dataset (e.g., using label studio like tools)
- Evaluate an off-the shelf solution if it exists (e.g., a cloud service provider)
- Create automatically labeled data and build a model using weak supervision, evaluating with your high quality test data.

Revisiting: Practical Advice-2

- You managed to get some labeled data through automatic labeling or other means.
- You also managed a baseline weakly supervised model.
- Then, what?

Revisiting: Practical Advice-2

- You managed to get some labeled data through automatic labeling or other means.
- You also managed a baseline weakly supervised model.
- Then, what?
- Evaluate transfer learning if a similar model is available
- Consider if Semi-supervised learning and/or Active learning will be useful

Revisiting: Practical Advice-3

- Slowly, you built up a large collection of labeled or pseudo-labeled data.
- You can then explore more sophisticated ML/DL models

Before all this, think if you really need all this, too. Rule based matching may just be sufficient for your scenario! (Check out spacy's rule based matching feature!)

Summary

- manual labeling to compile a high quality test dataset
- using off the shelf solutions, if available, and evaluating them
- using weak supervision, to build a labeled dataset automatically, and then training a ML/DL model
- how this compares with the case when we have some labeled dataset (regular classification, transfer learning)
- an overview of other methods: (semi-supervised, active learning)

Source code etc.

- Session5-Materials
- Some are in the form of a Notebook. Some are as .py files
- There may even be code snippets without the code file given.

- I leave exploring and making these work as exercises for future, so that you don't forget stuff! :D

Other useful articles

- Eugene Yan's blog post on bootstrapping labeled data
- transfer learning lesson from huggingface
- "A Survey on Recent Approaches for Natural Language Processing in Low-Resource Scenarios"

What more is happening?

- zero-shot transfer (where you don't have to retrain/finetune etc.
- few-shot learning (where you can learn with very small training data)
- prompt based learning/fine-tuning (where you give hints to the NLP model in plain text!)

etc. This is all ongoing work, and it will take a little of time to see whether they can be useful beyond NLP community, in other disciplines.

Tomorrow's class

- Schedule (TODO)