Matheus Schmitz
USC ID: 5039286453

Using Snorkel to Extract Performances and their Directors

Notes:

- **You are supposed to write your code or modify our code in any cell with `# TODO`.**
- **Much content of this notebook was borrowed from Snorkel Introduction Tutorial**

State-of-the-art extraction techniques require massive labeled training set but it is costly to obtain. To overcome this problem, Snorkel helps rapidly create training sets using the new data programming paradigm. To start, developers focus on writing a set of labeling functions, which are just scripts that programmatically label data. The resulting labels are noisy, but Snorkel uses a generative model to learn how to use those labeling functions to label more data. The new labeled data now can be used to train high-quality end models.

In summary, in this task, you will first manually label 50 documents and use these labeled data as a development set to create your own labeling functions. Then, you will train a generative model to label the rest 450 documents in training set. Finally, you will train a discriminative model (Bi-LSTM) to produce your final extraction model!

Task

In this homework, you need to extract the list of `performances` and their `directors` from the set of IMDB biographies that you collect for Homework 2. For example, you need to extract three tuples: [(`Lost on Purpose`, `the Nelms Brothers`), (`Waffle Street`, `the Nelms Brothers`), (`Small Town Crime`, `the Nelms Brothers`)] from the following sentence.

```
He would go on to act in three consecutive, but very different films written and directed by the Nelms Brothers: Lost on Purpose, Waffle Street and Small Town Crime.
```

In cases where your collected biographies do not contain enough pairs of `performances` and `directors`, please feel free to use the example dataset as well.

In [1]:

```
# TODO: COMBINE ALL OF YOUR BIOGRAPHIES IN ONE CSV FILE AND SUBMIT "Firstname_Lastname_hw05_all.csv"
import pandas as pd
import os
if not os.path.isfile('Matheus_Schmitz_hw05_all.tsv'):
    # Read the example dataset
    df1 = pd.read_csv('cast_bios.tsv', sep='\t', header=None)
    print(f'df1.shape: {df1.shape}')

    # Read my dataset from hw02
    df2 = pd.read_csv('Matheus_Schmitz_hw02_bios.csv', header=None)
    df2.to_csv('Matheus_Schmitz_hw05_bio.tsv', header=False, index=False, sep='\t')
    print(f'df2.shape: {df2.shape}')

    # Merge the datasets
    df_merged = pd.concat([df1, df2])
    print(f'df_merged.shape: {df_merged.shape}')

    # Deduplicate
    df_merged.drop_duplicates(inplace=True)
    print(f'deduplication: {df_merged.shape}')

    # Save as CSV and TSV
    df_merged.to_csv('Matheus_Schmitz_hw05_all.csv', header=False, index=False)
```

```
    df_merged.to_csv('Matheus_Schmitz_hw05_all.tsv', header=False, index=False, sep='\t'
)
```

Prepare environment

Lets install the packages we will use. Through my testing, Snorkel v0.7 works the best with Python 3.6

In [2]:

```
# If you are using Anaconda, you can create a new Python 3.6 environment.

# !conda create -n py36 python=3.6
```

In [3]:

```
#!pip install -r requirements.txt
```

We will work with Snorkel version 0.7 (Beta), we can retrieve it by running the following commands:

In [4]:

```
#!curl -L "https://github.com/snorkel-team/snorkel/archive/v0.7.0-beta.tar.gz" -o snorkel
_v0_7_0.tar.gz
```

Now let's uncompress the package and install Snorkel

In [5]:

```
#!tar -xvzf snorkel_v0_7_0.tar.gz
```

In [6]:

```
#!pip install snorkel-0.7.0-beta/
```

Creating a development set

We need to preprocess our documents using `Snorkel` utilities, parsing them into a simple hierarchy of component parts of our input data, which we refer as *contexts*. We'll also create *candidates* out of these contexts, which are the objects we want to classify, in this case, possible mentions of schools and colleges that the cast have attended. Finally, we'll load some gold labels for evaluation.

All of this preprocessed input data is saved to a database. In Snorkel, if no database is specified, then a SQLite database at `./snorkel.db` is created by default -- so no setup is needed here!

In [7]:

```
# ** STUDENT CODE

import numpy as np, os
from pathlib import Path

from snorkel import SnorkelSession
from snorkel.parser import TSVDocPreprocessor, CorpusParser, CSVPathsPreprocessor
from snorkel.parser.spacy_parser import Spacy
from snorkel.models import Document, Sentence, candidate_subclass
from snorkel.viewer import SentenceNgramViewer
from snorkel.annotations import LabelAnnotator, load_gold_labels

# TODO: SET LOCATION WHERE YOU STORE YOUR HW5 FILES
if 'HW_DIR' not in os.environ:
    HW_DIR = Path(".")
else:
    HW_DIR = Path(os.environ['HW_DIR'])
    assert HW_DIR.exists()
```

Initializing a `SnorkelSession`

In [8]:

```
%load_ext autoreload
%autoreload 2
%matplotlib inline

session = SnorkelSession()
```

Loading the Corpus

Next, we load and pre-process the corpus of documents.

In [9]:

```
# Using only hw2 dataset
doc_preprocessor = TSVDocPreprocessor(HW_DIR / 'Matheus_Schmitz_hw05_bio.tsv')
```

Running a `CorpusParser`

We'll use [Spacy](), an NLP preprocessing tool, to split our documents into sentences and tokens, and provide named entity annotations.

In [10]:

```
# Uncomment this to download spacy model
# !python -m spacy download [model_name] (e.g. en_core_web_lg)

#corpus_parser = CorpusParser(parser=Spacy())
#%time corpus_parser.apply(doc_preprocessor)
```

We can then use simple database queries (written in the syntax of [SQLAlchemy](), which Snorkel uses) to check how many documents and sentences were parsed:

In [11]:

```
print("Documents:", session.query(Document).count())
print("Sentences:", session.query(Sentence).count())
```

```
Documents: 982
Sentences: 4519
```

Generating Candidates

The next step is to extract *candidates* from our corpus. A `Candidate` in Snorkel is an object for which we want to make a prediction. In this case, the candidates are pairs of performances and directors mentioned in sentences.

The [Spacy]() parser we used performs *named entity recognition* for us. Next, we'll split up the documents into train and development splits; and collect the associated sentences.

Writing a simple director name matcher

Our simple name matcher makes use of the fact that the names of the directors are mentions of person-type named entities in the documents. `Fonduer` provides a list of built-in matchers that can be used in many information extraction tasks. We will use `PersonMatcher` to extract director names.

In [12]:

```
%%capture
```

```
from snorkel.matchers import PersonMatcher, OrganizationMatcher
from snorkel.matchers import RegexMatchEach, LambdaFunctionMatcher

director_matcher = PersonMatcher(longest_match_only=True)
```

2.1. Define (in the notebook) two matchers: one for performances and one for directors

In [13]:

```
# ** STUDENT CODE

# TODO: WRITE YOUR DIRECTOR MATCHER. YOU CAN REUSE EXTRACTORS IN HOMEWORK 2
import re

by_director = re.compile(r"(?<=by)((.|\n|\s|\S)*?)(?=\.|\,)")
directors_movie = re.compile(r"([a-zA-Z'-]+)(?=\S*['])")
director_regex = re.compile("director")
def has_director(mention):
    ner_tag = mention.get_attrib_span('ner_tags')
    if 'person' in ner_tag.lower():
        director_string = mention.get_span()
        m1 = by_director.findall(director_string) # by Director
        m2 = directors_movie.findall(director_string) # Directors's Movie
        m3 = director_regex.findall(director_string) # director Director
        matches = m1 + m2 + m3
        if len(matches) > 0:
            return True
        else:
            return False
    else:
        return False

director__matcher = LambdaFunctionMatcher(func=has_director)
```

Writing a random performance matcher

We design our random award matcher to capture all capitalized `span`s of text that contain the letter `A`.

In [14]:

```
# ** STUDENT CODE

# TODO: WRITE YOUR PERFORMANCE MATCHER. YOU CAN REUSE EXTRACTORS IN HOMEWORK 2
from snorkel.matchers import RegexMatchEach, LambdaFunctionMatcher

parethesis_year = re.compile(r"[A-Z][a-z]*\s\(\d{4}\)")
single_quotes_regex = re.compile(r"'[^']+'")
double_quotes_regex = re.compile(r'"[^\']+"')
directed_by_regex = re.compile('directed by')
def has_performance(mention):
    if 'direct' in mention.sentence.text:
        ner_tag = mention.get_attrib_span('ner_tags')
        if " O " in ner_tag and "ORG" not in ner_tag and "PERSON" not in ner_tag:
            performance_string = mention.get_span()
            m1 = parethesis_year.findall(performance_string) # (1234)
            m2 = double_quotes_regex.findall(performance_string) # "Movie Name"
            m3 = directed_by_regex.findall(performance_string) # directed by
            m4 = single_quotes_regex.findall(performance_string) # 'Movie Name'
            matches = m1 + m2 + m3
            if len(matches) > 0:
                return True
            else:
                return False
        else:
            return False
    else:
        return False
```

```
performance_matcher = LambdaFunctionMatcher(func=has_performance)
```

We know that normally each `director` name will contain at least two words (first name, last name). Considering additional middle names, we expect a maximum of four words per name.

Similarly, we assume the `performance` name to be a `span` of one to seven words.

We use the default `Ngrams` class provided by `Fonduer` to define these properties:

In [15]:

```
from snorkel.candidates import Ngrams
# ** STUDENT CODE

# TODO: FEEL FREE TO CHANGE THE NGRAMS LENGTH IF YOU WANT
performance_ngrams = Ngrams(n_max=7)
director_ngrams = Ngrams(n_max=4)
```

We create a candidate that is composed of a `performance` and a `director` mention as we defined above. We name this candidate `performance_director`. And we will extract all

In [16]:

```
from snorkel.candidates import Ngrams, CandidateExtractor

performance_with_director = candidate_subclass('performance_director', ['performance', 'd
irector'])
ngrams = Ngrams(n_max=7)
cand_extractor = CandidateExtractor(performance_with_director, [performance_ngrams, direc
tor_ngrams], [performance_matcher, director_matcher])
```

Create the development set

We create our development set by generating a `dev_ids.csv` file, which has one column `id` and contains 50 random biography URLs. You can choose any subset of 50 biographies that have `performance` and `director`.

In [17]:

```
docs = session.query(Document).order_by(Document.name).all()
import pandas as pd

docs = session.query(Document).order_by(Document.name).all()
ld = len(docs)

gold_data = pd.read_csv("dev_ids.csv")

dev_docs = gold_data["id"].values.tolist()

print(f"Number of dev documents: {len(dev_docs)}")

train_sents = set()
dev_sents   = set()

for doc in docs:
    sents = [s for s in doc.sentences]
    if doc.name in dev_docs:
        dev_sents.update(sents)
    else:
        train_sents.update(sents)

print("Number of dev sents:", len(dev_sents))
print("Number of train sents:", len(train_sents))
```

```
Number of dev documents: 50
Number of dev sents: 756
```

```
Number of train sents: 3763
```

Finally, we'll apply the candidate extractor to the two sets of sentences. The results will be persisted in the database backend.

In [18]:

```
%%time

for i, sents in enumerate([train_sents, dev_sents]):
    cand_extractor.apply(sents, split=i)
    print("Number of candidates:", session.query(performance_with_director).filter(perfo
rmance_with_director.split == i).count())
```

```
Clearing existing...
Running UDF...
[========================================] 100%

Number of candidates: 87
Clearing existing...
Running UDF...
[========================================] 100%

Number of candidates: 212
Wall time: 9.64 s
```

Label 50 documents in development set

In this task, you will use `SentenceNgramViewer` to label each mention. You can click the green button to mark the candidate as correct, red button to mark as incorrect. Your labeling result is automatically stored in the database.

In [19]:

```
from snorkel.models import GoldLabel, GoldLabelKey

def get_gold_labels(session: SnorkelSession, annotator_name: str="gold"):
    # define relationship in case it is not defined
    ak = session.query(GoldLabelKey).filter(GoldLabelKey.name == annotator_name).first()
    return session.query(GoldLabel).filter(GoldLabel.key == ak).all()

gold_labels = get_gold_labels(session)
labeled_sents = {lbl.candidate.performance.sentence.id for lbl in gold_labels}
unlabeled = [
    x for x in session.query(performance_with_director).filter(performance_with_director
.split == 1).all()
    if x.performance.sentence.id not in labeled_sents
]
print("Number unlabeled:", len(unlabeled))
```

```
Number unlabeled: 212
```

Please remember to label all pairs of mentions, both correct and incorrect ones

`SentenceNgramViewer` only show candidates that are matched by your matchers. Therefore, your annotation is under an assumption that your matchers work perfectly.

In [20]:

```
# Uncomment and run this if you see "SentenceNgramViewer" text instead of a UI component.
Then restart your notebook and refresh your browser.

#!jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

In [21]:

```
SentenceNgramViewer(unlabeled, session, annotator_name="gold")
```

After you finish labeling, executing the cell below to save your result to CSV files.

In [22]:

```
# ** STUDENT CODE

def extract_gold_labels(session: SnorkelSession, annotator_name: str="gold", split: int=N
one):
    ''' Extract pairwise gold labels and store in a file. '''
    gold_labels = get_gold_labels(session, annotator_name)

    results = []
    for gold_label in gold_labels:
        rel = gold_label.candidate
        if split is not None and rel.split != split:
            continue

        results.append({
            "id": rel.performance.sentence.document.name,
            "performance": rel.performance.get_span(),
            "director": rel.director.get_span(),
            "value": gold_label.value
        })

    return results

#gold_labels = extract_gold_labels(session, split=1)
#gold_labels
```

In [23]:

```
# TODO: CHANGE TO YOUR NAME AND SAVE THE GOLD LABELS (TASK 1)
#pd.DataFrame(gold_labels).to_csv("Matheus_Schmitz_hw05_gold.dev.csv", index=None)
```

In [24]:

```
# ** STUDENT CODE
import json
from snorkel.models import StableLabel
from snorkel.db_helpers import reload_annotator_labels

def save_gold_labels(session: SnorkelSession, annotator_name: str="gold", split: int=None
, output_file="saved_gold.json"):
    ''' Extract pairwise gold labels and store in a file. '''
    gold_labels = get_gold_labels(session, annotator_name)

    results = []
    for gold_label in gold_labels:
        rel = gold_label.candidate
        if split is not None and rel.split != split:
            continue

        results.append({
            "performance": rel.performance.stable_id,
            "director": rel.director.stable_id,
            "value": gold_label.value
        })

    with open(str(output_file), "w") as f:
        json.dump(results, f, indent=4)

#save_gold_labels(session, "gold", split=1, output_file="saved_gold.json")
```

In [25]:

```
def reload_external_labels(session: SnorkelSession, input_file, annotator_name: str="gold
", split: int=None):
    performance_with_director = candidate_subclass('performance_director', ['performance'
```

```
, 'director'])
    with open(str(input_file), "r") as f:
        lbls = json.load(f)

    for lbl in lbls:
        # we check if the label already exists, in case this cell was already executed
        context_stable_ids = "~~".join((lbl['performance'], lbl['director']))
        query = session.query(StableLabel).filter(StableLabel.context_stable_ids == cont
ext_stable_ids)
        query = query.filter(StableLabel.annotator_name == annotator_name)
        if query.count() == 0:
            session.add(StableLabel(
                context_stable_ids=context_stable_ids,
                annotator_name=annotator_name,
                value=lbl['value']
            ))

    # commit session
    session.commit()

    # reload annotator labels
    reload_annotator_labels(session, performance_with_director, annotator_name, split=spl
it, filter_label_split=False)

reload_external_labels(session, "saved_gold.json", "gold", split=1)
```

```
AnnotatorLabels created: 212
```

Define labeling functions (LFs)

In this task, you will define your own LFs, which Snorkel uses to create noise-aware training set. Usually, you will go through a couple of iterations (create LFs, test and refine it) to come up with a good set of LFs. We provide you at the end of this section a helper to quickly see what candidates did your model fail to classify. You can refer to [Snorkel tutorial] for more information.

You are free to use write any extra code to create a set of sophisticated LFs. More LF helper functions can be found [here].

2.2. Define (in the notebook) at least three labeling functions

In [26]:

```
# ** STUDENT CODE

# THESE ARE SOME HELPER FUNCTIONS THAT YOU CAN USE
from snorkel.lf_helpers import (
    get_left_tokens, get_right_tokens, get_between_tokens,
    get_text_between, get_tagged_text,
)

# TODO: DEFINE YOUR LFS HERE. BELOW ARE SOME RANDOM LFS

FALSE = -1
ABSTAIN = 0
TRUE = 1

re1 = re.compile(r"[A-Z][a-z]*\s[A-Z][a-z]*")
def LF_2_capitalized_seq_P(c):
    match = re1.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return FALSE
def LF_2_capitalized_seq_D(c):
    match = re1.findall(c.director.get_span())
    if len(match) > 0:
        return TRUE
```

```
    else:
        return FALSE

re2 = re.compile(r"[A-Z][a-z]*\s[a-z]*\s[A-Z][a-z]*")
def LF_upper_lower_upper_P(c):
    match = re2.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN

re4 = re.compile(r'"[^\']+"')
def LF_double_quotes_P(c):
    match = re4.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN
def LF_double_quotes_D(c):
    match = re4.findall(c.director.get_span())
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN

re6 = re.compile(r"([a-zA-Z']+)(?=\S*['])")
def LF_apostrophe_D(c):
    match = re6.findall(c.director.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN

re7 = re.compile(r"'[^']+'")
def LF_single_quotes_P(c):
    match = re7.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN
def LF_single_quotes_D(c):
    match = re7.findall(c.director.get_span())
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN

re3 = re.compile(r"[A-Z][a-z]*\s\(\d{4}\)")
def LF_year_P(c):
    match = re3.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN

def LF_quote_or_year(c):
    m1 = re7.findall(c.performance.get_span())
    m2 = re3.findall(c.performance.get_span())
    match = m1 + m2
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN

re10 = re.compile(r"(?<=\')[\w\s]+(?=\.|\,)")
def LF_possesive_P(c):
    match = re10.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN
def LF_possesive_D(c):
```

```
    match = re10.findall(c.director.get_span())
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN


re11 = re.compile(r"[A-Z][a-z]*\s[A-Z][a-z]*\s[A-Z][a-z]*")
def LF_3_capitalized_seq_P(c):
    match = re11.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN
def LF_3_capitalized_seq_D(c):
    match = re11.findall(c.director.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN


re12 = re.compile(r"[A-Z][a-z]*\s[A-Z][a-z]*'")
def LF_2_titlecase_apostrophe_D(c):
    match = re12.findall(c.director.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN


re13 = re.compile(r"[A-Z][a-z]*'")
def LF_1_titlecase_apostrophe_D(c):
    match = re13.findall(c.director.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN
def LF_1_titlecase_apostrophe_P(c):
    match = re13.findall(c.performance.get_span())
    if len(match) > 0:
        return ABSTAIN
    else:
        return ABSTAIN


re15 = re.compile(r"[A-Z][a-z][a-z]+")
def LF_titlecase_P(c):
    match = re15.findall(c.performance.get_span())
    if len(match) > 0:
        return ABSTAIN
    else:
        return FALSE


re16 = re.compile(r"[A-Z][a-z][a-z]+\s[A-Z][a-z][a-z]+")
def LF_titlecase_D(c):
    match = re16.findall(c.director.get_span())
    if len(match) > 0:
        return ABSTAIN
    else:
        return FALSE


re14 = re.compile(r"\d")
def LF_digit_P(c):
    match = re14.findall(c.performance.get_span())
    if len(match) > 0:
        return TRUE
    else:
        return ABSTAIN
```

```python
re17 = re.compile("directed by")
re15 = re.compile(r"[A-Z][a-z][a-z]+")
def LF_movie_direct_P(c):
    m1 = re17.findall(c.performance.get_span())
    m2 = re15.findall(c.performance.get_span())
    if len(m1) > 0 and len(m2) == 0:
        return FALSE
    else:
        return ABSTAIN


re18 = re.compile("comedy")
re19 = re.compile("drama")
re20 = re.compile("horror")
re21 = re.compile("reuniting")
re22 = re.compile("romance")
re23 = re.compile("film")
re24 = re.compile("award")
def LF_genre_D(c):
    m1 = re18.findall(c.director.get_span())
    m2 = re19.findall(c.director.get_span())
    m3 = re20.findall(c.director.get_span())
    m4 = re21.findall(c.director.get_span())
    m5 = re22.findall(c.director.get_span())
    m6 = re23.findall(c.director.get_span())
    m7 = re24.findall(c.director.get_span())
    match = m1 + m2 + m3 + m4 + m5 + m6 + m7
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN
def LF_genre_P(c):
    m1 = re18.findall(c.performance.get_span())
    m2 = re19.findall(c.performance.get_span())
    m3 = re20.findall(c.performance.get_span())
    m4 = re21.findall(c.performance.get_span())
    m5 = re22.findall(c.performance.get_span())
    m6 = re23.findall(c.performance.get_span())
    m7 = re24.findall(c.performance.get_span())
    match = m1 + m2 + m3 + m4 + m5 + m6 + m7
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN

re26 = re.compile('act')
def LF_actors_P(c):
    match = re26.findall(c.performance.get_span())
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN



def LF_first_capitalized_D(c):
    string = c.director.get_span()
    if string[0].islower():
        return FALSE
    else:
        return ABSTAIN
def LF_first_capitalized_P(c):
    string = c.performance.get_span()
    if string[0].islower():
        return FALSE
    else:
        return ABSTAIN

def LF_nearby_5(c):
    if len(list(get_between_tokens(c))) < 5:
        return TRUE
```

```
    else:
        return ABSTAIN

def LF_directed_by(c):
    bet_text = get_text_between(c)
    if "directed" in bet_text:
        return TRUE
    else:
        return ABSTAIN

def LF_parenthesis_right(c):
    right_tokens = get_right_tokens(c)
    if "(" in ' '.join(right_tokens):
        return TRUE
    else:
        return ABSTAIN

regex_yb = re.compile(r"\d{4}")
def LF_year_between(c):
    bet_tokens = ' '.join(get_between_tokens(c))
    match = regex_yb.findall(bet_tokens)
    if match:
        return FALSE
    else:
        return ABSTAIN

def LF_star(c):
    bet_tokens = get_between_tokens(c)
    if "star" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_act(c):
    bet_tokens = get_between_tokens(c)
    if "act" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_with(c):
    bet_tokens = get_between_tokens(c)
    if "with" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_feat(c):
    bet_tokens = get_between_tokens(c)
    if "feat" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_featuring(c):
    bet_tokens = get_between_tokens(c)
    if "featuring" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_opposite(c):
    bet_tokens = get_between_tokens(c)
    if "opposite" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN


def LF_which(c):
    bet_tokens = get_between_tokens(c)
    if "which" in ' '.join(bet_tokens):
```

```
        return FALSE
    else:
        return ABSTAIN

def LF_film(c):
    bet_tokens = get_between_tokens(c)
    if "film" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_work(c):
    bet_tokens = get_between_tokens(c)
    if "work" in ' '.join(bet_tokens):
        return FALSE
    else:
        return ABSTAIN

def LF_based(c):
    bet_text = get_text_between(c)
    if "based" in bet_text or "earlier" in bet_text:
        return FALSE
    else:
        return ABSTAIN

def LF_drama(c):
    bet_text = get_text_between(c)
    if "drama" in bet_text:
        return FALSE
    else:
        return ABSTAIN

def LF_nominat(c):
    bet_text = get_text_between(c)
    if "nominat" in bet_text or "feat" in bet_text:
        return FALSE
    else:
        return ABSTAIN

def LF_final(c):
    bet_text = get_text_between(c)
    if "final" in bet_text:
        return FALSE
    else:
        return ABSTAIN

def LF_reunit(c):
    bet_text = get_text_between(c)
    if "reunit" in bet_text:
        return FALSE
    else:
        return ABSTAIN

def LF_document(c):
    bet_text = get_text_between(c)
    if "document" in bet_text or "film" in bet_text:
        return FALSE
    else:
        return ABSTAIN

re15 = re.compile(r"[A-Z][a-z][a-z]+")
def LF_names_between(c):
    bet_text = get_text_between(c)
    match = re15.findall(bet_text)
    if len(match) >= 3:
        return FALSE
    else:
        return ABSTAIN

re15 = re.compile(r"[A-Z][a-z][a-z]+")
def LF_partners(c):
    bet_text = get_text_between(c)
```

```
    match = re15.findall(bet_text)
    if len(match) >= 3 and 'opposite' in bet_text:
        return FALSE
    else:
        return ABSTAIN

re3 = re.compile(r"[A-Z][a-z]*\s\(\d{4}\)")
def LF_movie_between(c):
    bet_text = get_text_between(c)
    match = re3.findall(bet_text)
    if len(match) > 0:
        return FALSE
    else:
        return ABSTAIN

re25 = re.compile(",")
def LF_commas_between(c):
    bet_text = get_text_between(c)
    match = re25.findall(bet_text)
    if len(match) >= 2:
        return FALSE
    else:
        return ABSTAIN

def LF_relations(c):
    bet_text = get_text_between(c)
    if 'and the' in bet_text or 'and The' in bet_text or 'was a' in bet_text:
        return FALSE
    else:
        return ABSTAIN
```

In [27]:

```
# ** STUDENT CODE

# TODO: PUT ALL YOUR LABELING FUNCTIONS HERE

performance_with_director_lfs = [
    LF_upper_lower_upper_P,
    LF_year_P,  LF_single_quotes_P, LF_single_quotes_D,
    LF_1_titlecase_apostrophe_D, LF_1_titlecase_apostrophe_P, LF_digit_P,
    LF_directed_by, LF_parenthesis_right, LF_year_between,
    LF_star, LF_act, LF_with, LF_commas_between,
    LF_feat, LF_featuring, LF_opposite,
    LF_titlecase_P, LF_titlecase_D,
    LF_which, LF_film, LF_work, LF_based, LF_quote_or_year,
    LF_movie_direct_P, LF_genre_D, LF_genre_P,
    LF_drama, LF_nominat, LF_final, LF_reunit, LF_document,
    LF_first_capitalized_D, LF_first_capitalized_P, LF_names_between,
    LF_partners, LF_movie_between,
    LF_actors_P, LF_relations,
]
```

Train generative model

Now, we'll train a model of the LFs to estimate their accuracies. Once the model is trained, we can combine the outputs of the LFs into a single, noise-aware training label set for our extractor. Intuitively, we'll model the LFs by observing how they overlap and conflict with each other.

In [28]:

```
np.random.seed(1701)

labeler = LabelAnnotator(lfs=performance_with_director_lfs)
L_train = labeler.apply(split=0)
```

```
Clearing existing...
Running UDF...
[========================================] 100%
```

Get detailed statistics of LFs before training the model

2.3. Report the performance of your LFs before generative model training

In [29]:

```
L_train.lf_stats(session)
```

Out[29]:

	j	Coverage	Overlaps	Conflicts
LF_upper_lower_upper_P	0	0.103448	0.103448	0.103448
LF_year_P	1	0.379310	0.379310	0.367816
LF_single_quotes_P	2	0.000000	0.000000	0.000000
LF_single_quotes_D	3	0.000000	0.000000	0.000000
LF_1_titlecase_apostrophe_D	4	0.000000	0.000000	0.000000
LF_1_titlecase_apostrophe_P	5	0.000000	0.000000	0.000000
LF_digit_P	6	0.390805	0.390805	0.379310
LF_directed_by	7	0.011494	0.011494	0.011494
LF_parenthesis_right	8	0.425287	0.425287	0.413793
LF_year_between	9	0.287356	0.287356	0.287356
LF_star	10	0.000000	0.000000	0.000000
LF_act	11	0.000000	0.000000	0.000000
LF_with	12	0.011494	0.011494	0.011494
LF_commas_between	13	0.344828	0.344828	0.103448
LF_feat	14	0.011494	0.011494	0.011494
LF_featuring	15	0.000000	0.000000	0.000000
LF_opposite	16	0.000000	0.000000	0.000000
LF_titlecase_P	17	0.000000	0.000000	0.000000
LF_titlecase_D	18	0.597701	0.574713	0.459770
LF_which	19	0.000000	0.000000	0.000000
LF_film	20	0.149425	0.149425	0.080460
LF_work	21	0.011494	0.011494	0.011494
LF_based	22	0.000000	0.000000	0.000000
LF_quote_or_year	23	0.379310	0.379310	0.367816
LF_movie_direct_P	24	0.000000	0.000000	0.000000
LF_genre_D	25	0.000000	0.000000	0.000000
LF_genre_P	26	0.068966	0.068966	0.034483
LF_drama	27	0.000000	0.000000	0.000000
LF_nominat	28	0.011494	0.011494	0.011494
LF_final	29	0.000000	0.000000	0.000000
LF_reunit	30	0.000000	0.000000	0.000000
LF_document	31	0.149425	0.149425	0.080460
LF_first_capitalized_D	32	0.000000	0.000000	0.000000
LF_first_capitalized_P	33	0.310345	0.298851	0.183908
LF_names_between	34	0.367816	0.344828	0.103448
LF_partners	35	0.000000	0.000000	0.000000

	j	Coverage	Overlaps	Conflicts
LF_movie_between	36	0.045977	0.045977	0.045977
LF_actors_P	37	0.022989	0.022989	0.022989
LF_relations	38	0.000000	0.000000	0.000000

In [30]:

```
# TODO: MAKE SURE THE ABOVE CELL OUTPUT IS SHOWN IN YOUR PDF VERSION. THIS WILL BE YOUR A
NSWER FOR TASK 2.3
```

Report the weights of your LFs after generative model training

In [31]:

```
from snorkel.learning import GenerativeModel

gen_model = GenerativeModel()
gen_model.train(L_train, epochs=100, decay=0.95, step_size=0.1 / L_train.shape[0], reg_p
aram=1e-6)

print("LF weights:", gen_model.weights.lf_accuracy)
```

```
Inferred cardinality: 2
LF weights: [ 0.19427958  0.6262435   0.06462321  0.0473138   0.07642905  0.092425
  0.62782561  0.07128648  0.4226728  -0.04638931  0.08068634  0.06897329
  0.06805461  0.52322153  0.08839914  0.07982269  0.07711804  0.07006701
 -0.20495203  0.09041777  0.21279293  0.07880266  0.08296787  0.6146284
  0.06758788  0.06471318  0.1371921   0.0850499   0.09235736  0.0805227
  0.05463209  0.20287494  0.10353593  0.02409021  0.54250966  0.07087675
 -0.03227999  0.04385748  0.09728198]
```

In [32]:

```
# TODO: MAKE SURE THE ABOVE CELL OUTPUT IS SHOWN IN YOUR PDF VERSION. THIS WILL BE YOUR A
NSWER FOR TASK 2.2
```

Now that we have learned the generative model, we will measure its performances using the provided test set

In [33]:

```
L_gold_dev = load_gold_labels(session, annotator_name='gold', split=1)
```

In [34]:

```
L_dev = labeler.apply_existing(split=1)
tp, fp, tn, fn = gen_model.error_analysis(session, L_dev, L_gold_dev)
```

```
Clearing existing...
Running UDF...
[========================================] 100%

========================================
Scores (Un-adjusted)
========================================
Pos. class accuracy: 0.558
Neg. class accuracy: 0.677
Precision            0.796
Recall               0.558
F1                   0.656
----------------------------------------
TP: 82 | FP: 21 | TN: 44 | FN: 65
========================================
```

Get detailed statistics of LFs learned by the model

Report the performance of your LFs after generative model training

Report the performance of your LFs after generative model training

In [35]:

```
L_dev.lf_stats(session, L_gold_dev, gen_model.learned_lf_stats()['Accuracy'])
```

```
C:\Users\Matheus\Anaconda3\lib\site-packages\snorkel\annotations.py:137: RuntimeWarning:
invalid value encountered in true_divide
  ac = (tp+tn) / (tp+tn+fp+fn)
```

Out[35]:

	j	Coverage	Overlaps	Conflicts	TP	FP	FN	TN	Empirical Acc.	Learned Acc.
LF_upper_lower_upper_P	0	0.047170	0.023585	0.023585	10	0	0	0	1.000000	0.587151
LF_year_P	1	0.216981	0.216981	0.174528	30	16	0	0	0.652174	0.776944
LF_single_quotes_P	2	0.084906	0.084906	0.066038	15	3	0	0	0.833333	0.530216
LF_single_quotes_D	3	0.000000	0.000000	0.000000	0	0	0	0	NaN	0.536373
LF_1_titlecase_apostrophe_D	4	0.099057	0.094340	0.089623	17	4	0	0	0.809524	0.527148
LF_1_titlecase_apostrophe_P	5	0.000000	0.000000	0.000000	0	0	0	0	NaN	0.541905
LF_digit_P	6	0.297170	0.297170	0.245283	47	16	0	0	0.746032	0.772901
LF_directed_by	7	0.325472	0.311321	0.297170	55	14	0	0	0.797101	0.537723
LF_parenthesis_right	8	0.136792	0.117925	0.108491	24	5	0	0	0.827586	0.700797
LF_year_between	9	0.193396	0.193396	0.174528	0	0	30	11	0.268293	0.475921
LF_star	10	0.009434	0.009434	0.009434	0	0	2	0	0.000000	0.535799
LF_act	11	0.047170	0.037736	0.014151	0	0	1	9	0.900000	0.528386
LF_with	12	0.056604	0.056604	0.028302	0	0	3	9	0.750000	0.540357
LF_commas_between	13	0.207547	0.207547	0.202830	0	0	29	15	0.340909	0.736872
LF_feat	14	0.056604	0.056604	0.037736	0	0	1	11	0.916667	0.545590
LF_featuring	15	0.047170	0.047170	0.028302	0	0	0	10	1.000000	0.542001
LF_opposite	16	0.037736	0.037736	0.028302	0	0	0	8	1.000000	0.547173
LF_titlecase_P	17	0.146226	0.146226	0.080189	0	0	21	10	0.322581	0.538843
LF_titlecase_D	18	0.212264	0.212264	0.169811	0	0	32	13	0.288889	0.401810
LF_which	19	0.014151	0.014151	0.009434	0	0	0	3	1.000000	0.546456
LF_film	20	0.009434	0.009434	0.000000	0	0	0	2	1.000000	0.601021
LF_work	21	0.018868	0.018868	0.000000	0	0	0	4	1.000000	0.539415
LF_based	22	0.023585	0.023585	0.023585	0	0	3	2	0.400000	0.536509
LF_quote_or_year	23	0.301887	0.301887	0.240566	45	19	0	0	0.703125	0.776810
LF_movie_direct_P	24	0.099057	0.099057	0.042453	0	0	11	10	0.476190	0.532185
LF_genre_D	25	0.000000	0.000000	0.000000	0	0	0	0	NaN	0.529315
LF_genre_P	26	0.221698	0.221698	0.146226	0	0	17	30	0.638298	0.572276
LF_drama	27	0.028302	0.028302	0.028302	0	0	2	4	0.666667	0.548436
LF_nominat	28	0.056604	0.056604	0.037736	0	0	1	11	0.916667	0.551206
LF_final	29	0.014151	0.014151	0.014151	0	0	0	3	1.000000	0.543385
LF_reunit	30	0.018868	0.018868	0.018868	0	0	2	2	0.500000	0.523282
LF_document	31	0.018868	0.018868	0.009434	0	0	1	3	0.750000	0.606484
LF_first_capitalized_D	32	0.009434	0.009434	0.000000	0	0	2	0	0.000000	0.547261
LF_first_capitalized_P	33	0.466981	0.452830	0.367925	0	0	65	34	0.343434	0.512062
LF_names_between	34	0.367925	0.363208	0.330189	0	0	52	26	0.333333	0.752292
LF_partners	35	0.037736	0.037736	0.028302	0	0	0	8	1.000000	0.541138
LF_movie_between	36	0.047170	0.047170	0.047170	0	0	4	6	0.600000	0.484490

	j	Coverage	Overlaps	Conflicts	TP	FP	FN	TN	Empirical Acc.	Learned Acc.
LF_actors_P	37	0.014151	0.014151	0.004717	0	0	1	2	0.666667	0.518198
LF_relations	38	0.070755	0.070755	0.070755	0	0	2	13	0.866667	0.552162

In [36]:

```
# TODO: MAKE SURE THE ABOVE CELL OUTPUT IS SHOWN IN YOUR PDF VERSION. THIS WILL BE YOUR A
NSWER FOR TASK 2.3
```

We now apply the generative model to the training candidates to get the noise-aware training label set. We'll refer to these as the training marginals:

In [37]:

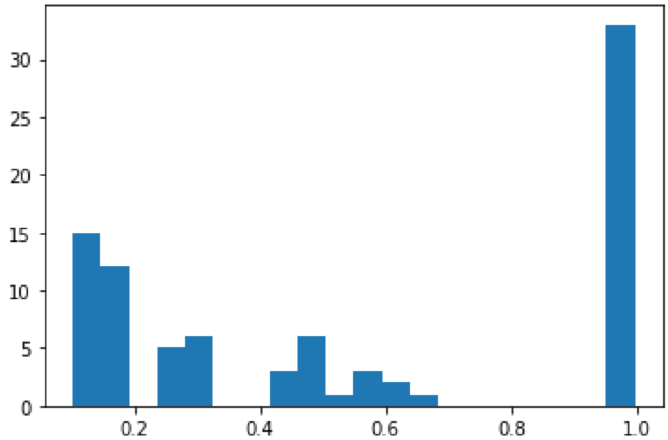
```
train_marginals = gen_model.marginals(L_train)
```

We'll look at the distribution of the training marginals:

2.4. Report the distribution of the training marginals

In [38]:

```
import matplotlib.pyplot as plt
plt.hist(train_marginals, bins=20)
plt.show()
```



In [39]:

```
# TODO: MAKE SURE THE ABOVE CELL OUTPUT IS SHOWN IN YOUR PDF VERSION. THIS WILL BE YOUR A
NSWER FOR TASK 2.4
```

In [40]:

```
# TODO: CHANGE THIS CELL TO MARKDOWN CELL AND WRITE YOUR ANSWER TO TASK 2.5 HERE.
```

2.5. Explain (in your notebook) about your marginal distribution (max 3 sentences). Is it good or bad? Explain briefly.

The distribution seems good, given that most classifiers are very close to either 0 or 1, as while it was initially very bad, but as I kept adding more labeling functions based on the FPs and FNs the distribution began to improve. The current distribution differentiates well between the classes, although its clear from the plot that defining what is a match is much easier than defining what is NOT a match.

You might want to look at some examples in one of the error buckets to improve your LFs. For example, below is one of the false positives that we did not correctly label correctly

In [41]:

```
SentenceNgramViewer(fn, session)
```

3. Adding Distant Supervision Labeling Function

Distant supervision generates training data automatically using an external, imperfectly aligned training resource, such as a Knowledge Base.

Define an additional distant-supervision-based labeling function which uses Wikidata or DBpedia. With the additional labeling function you added, please make sure to answer all questions for Task 3.3, 3.4, 3.5 mentioned in the homework.

In [42]:

```
from SPARQLWrapper import SPARQLWrapper, JSON

titlecase_regex = re.compile('[A-Z][a-z][a-z]+')
def LF_distant_supervision(c):

    performance_title = c.performance.get_span()
    performance_strings = ' '.join(titlecase_regex.findall(performance_title))
    director_name = c.director.get_span()
    director_strings = ' '.join(titlecase_regex.findall(director_name))

    sparql = SPARQLWrapper("http://dbpedia.org/sparql")
    sparql.setQuery(f'''
        PREFIX dcterms: <http://purl.org/dc/terms/>
        PREFIX foaf: <http://xmlns.com/foaf/0.1/>
        PREFIX owl: <http://www.w3.org/2002/07/owl#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX xml: <http://www.w3.org/XML/1998/namespace/>
        PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
        PREFIX dbo: <http://dbpedia.org/ontology/>
        prefix dbp: <http://dbpedia.org/property/>

        SELECT ?movie ?movieTitle ?directorName
        WHERE {{
        ?movie a dbo:Film .
        ?movie foaf:name ?movieTitle .
        ?movie dbp:director ?director .
        ?director foaf:name ?directorName .

        FILTER(CONTAINS(?movieTitle, "{performance_strings}"))
        FILTER(CONTAINS(?directorName , "{director_strings}"))
    }}
        ''')
    sparql.setReturnFormat(JSON)
    try:
        results = sparql.query().convert()
        binding = results['results']['bindings']
        movieTitle = binding[0]['movieTitle']['value']
        directorName = binding[0]['directorName']['value']
        match = True
    except:
        match = False
    if match:
        return TRUE
    else:
        return ABSTAIN
```

In [43]:

```
# ** STUDENT CODE

# TODO: PUT ALL YOUR LABELING FUNCTIONS HERE

performance_with_director_lfs = [
    LF_upper_lower_upper_P,
    LF_year_P,  LF_single_quotes_P, LF_single_quotes_D,
```

```
    LF_1_titlecase_apostrophe_D, LF_1_titlecase_apostrophe_P, LF_digit_P,
    LF_directed_by, LF_parenthesis_right, LF_year_between,
    LF_star, LF_act, LF_with, LF_commas_between,
    LF_feat, LF_featuring, LF_opposite,
    LF_titlecase_P, LF_titlecase_D,
    LF_which, LF_film, LF_work, LF_based, LF_quote_or_year,
    LF_movie_direct_P, LF_genre_D, LF_genre_P,
    LF_drama, LF_nominat, LF_final, LF_reunit, LF_document,
    LF_first_capitalized_D, LF_first_capitalized_P, LF_names_between,
    LF_partners, LF_movie_between,
    LF_actors_P, LF_relations,
    LF_distant_supervision
]
```

3.3 Report the performance of your LFs before generative model training

In [44]:

```
# Pre-Train
np.random.seed(1701)
labeler = LabelAnnotator(lfs=performance_with_director_lfs)
L_train = labeler.apply(split=0)
L_train.lf_stats(session)
```

```
Clearing existing...
Running UDF...
[========================================] 100%
```

Out[44]:

	j	Coverage	Overlaps	Conflicts
LF_upper_lower_upper_P	0	0.103448	0.103448	0.103448
LF_year_P	1	0.379310	0.379310	0.367816
LF_single_quotes_P	2	0.000000	0.000000	0.000000
LF_single_quotes_D	3	0.000000	0.000000	0.000000
LF_1_titlecase_apostrophe_D	4	0.000000	0.000000	0.000000
LF_1_titlecase_apostrophe_P	5	0.000000	0.000000	0.000000
LF_digit_P	6	0.390805	0.390805	0.379310
LF_directed_by	7	0.011494	0.011494	0.011494
LF_parenthesis_right	8	0.425287	0.425287	0.413793
LF_year_between	9	0.287356	0.287356	0.287356
LF_star	10	0.000000	0.000000	0.000000
LF_act	11	0.000000	0.000000	0.000000
LF_with	12	0.011494	0.011494	0.011494
LF_commas_between	13	0.344828	0.344828	0.103448
LF_feat	14	0.011494	0.011494	0.011494
LF_featuring	15	0.000000	0.000000	0.000000
LF_opposite	16	0.000000	0.000000	0.000000
LF_titlecase_P	17	0.000000	0.000000	0.000000
LF_titlecase_D	18	0.597701	0.574713	0.459770
LF_which	19	0.000000	0.000000	0.000000
LF_film	20	0.149425	0.149425	0.080460
LF_work	21	0.011494	0.011494	0.011494
LF_based	22	0.000000	0.000000	0.000000
LF_quote_or_year	23	0.379310	0.379310	0.367816

	j	Coverage	Overlaps	Conflicts
LF_movie_direct_P	24	0.000000	0.000000	0.000000
LF_genre_D	25	0.000000	0.000000	0.000000
LF_genre_P	26	0.068966	0.068966	0.034483
LF_drama	27	0.000000	0.000000	0.000000
LF_nominat	28	0.011494	0.011494	0.011494
LF_final	29	0.000000	0.000000	0.000000
LF_reunit	30	0.000000	0.000000	0.000000
LF_document	31	0.149425	0.149425	0.080460
LF_first_capitalized_D	32	0.000000	0.000000	0.000000
LF_first_capitalized_P	33	0.310345	0.298851	0.183908
LF_names_between	34	0.367816	0.344828	0.103448
LF_partners	35	0.000000	0.000000	0.000000
LF_movie_between	36	0.045977	0.045977	0.045977
LF_actors_P	37	0.022989	0.022989	0.022989
LF_relations	38	0.000000	0.000000	0.000000
LF_distant_supervision	39	0.000000	0.000000	0.000000

In [45]:

```
# Post-Train
gen_model = GenerativeModel()
gen_model.train(L_train, epochs=100, decay=0.95, step_size=0.1 / L_train.shape[0], reg_p
aram=1e-6)
print("LF weights:", gen_model.weights.lf_accuracy)
L_gold_dev = load_gold_labels(session, annotator_name='gold', split=1)
L_dev = labeler.apply_existing(split=1)
tp, fp, tn, fn = gen_model.error_analysis(session, L_dev, L_gold_dev)
L_dev.lf_stats(session, L_gold_dev, gen_model.learned_lf_stats()['Accuracy'])
```

```
Inferred cardinality: 2
LF weights: [ 0.19934435  0.61384383  0.06455938  0.04292462  0.08248233  0.08152397
  0.58449047  0.0616631   0.42670369 -0.0365053   0.06014309  0.09214489
  0.09174504  0.51274644  0.07547928  0.05926433  0.07285599  0.07276436
 -0.16940715  0.05422622  0.24344786  0.06373718  0.07257566  0.63409584
  0.08521251  0.07471332  0.11782195  0.0683143   0.08196633  0.05047449
  0.0815411   0.23043219  0.07798068  0.00841972  0.5565986   0.08851671
  0.03295081  0.04766071  0.06534586  0.05237717]
Clearing existing...
Running UDF...
[========================================] 100%

========================================
Scores (Un-adjusted)
========================================
Pos. class accuracy: 0.565
Neg. class accuracy: 0.677
Precision            0.798
Recall               0.565
F1                   0.661
----------------------------------------
TP: 83 | FP: 21 | TN: 44 | FN: 64
========================================
```

```
C:\Users\Matheus\Anaconda3\lib\site-packages\snorkel\annotations.py:137: RuntimeWarning:
invalid value encountered in true_divide
  ac = (tp+tn) / (tp+tn+fp+fn)
```

Out[45]:

	j	Coverage	Overlaps	Conflicts	TP	FP	FN	TN	Empirical Acc.	Learned Acc.

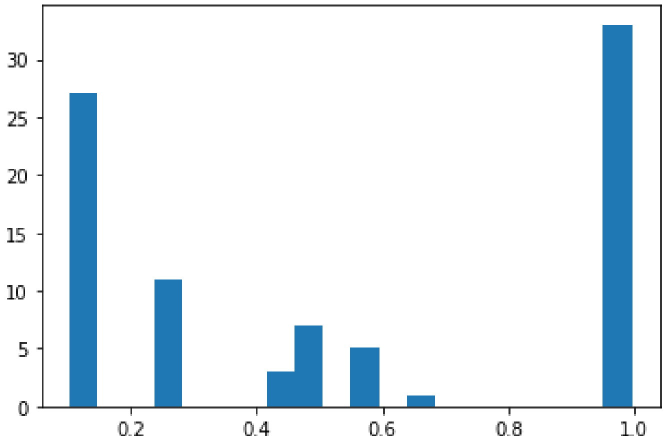
	j	Coverage	Overlaps	Conflicts	TP	FP	FN	TN	Empirical Acc.	Learned Acc.
LF_upper_lower_upper_P	0	0.047170	0.023585	0.023585	10	0	0	0	1.000000	0.590970
LF_year_P	1	0.216981	0.216981	0.174528	30	16	0	0	0.652174	0.772611
LF_single_quotes_P	2	0.084906	0.084906	0.066038	15	3	0	0	0.833333	0.541915
LF_single_quotes_D	3	0.000000	0.000000	0.000000	0	0	0	0	NaN	0.526420
LF_1_titlecase_apostrophe_D	4	0.099057	0.094340	0.089623	17	4	0	0	0.809524	0.537210
LF_1_titlecase_apostrophe_P	5	0.000000	0.000000	0.000000	0	0	0	0	NaN	0.546671
LF_digit_P	6	0.297170	0.297170	0.245283	47	16	0	0	0.746032	0.764621
LF_directed_by	7	0.325472	0.311321	0.297170	55	14	0	0	0.797101	0.538139
LF_parenthesis_right	8	0.136792	0.117925	0.108491	24	5	0	0	0.827586	0.702266
LF_year_between	9	0.193396	0.193396	0.174528	0	0	30	11	0.268293	0.481129
LF_star	10	0.009434	0.009434	0.009434	0	0	2	0	0.000000	0.522619
LF_act	11	0.047170	0.037736	0.014151	0	0	1	9	0.900000	0.553109
LF_with	12	0.056604	0.056604	0.028302	0	0	3	9	0.750000	0.551020
LF_commas_between	13	0.207547	0.207547	0.202830	0	0	29	15	0.340909	0.743442
LF_feat	14	0.056604	0.056604	0.037736	0	0	1	11	0.916667	0.533938
LF_featuring	15	0.047170	0.047170	0.028302	0	0	0	10	1.000000	0.533858
LF_opposite	16	0.037736	0.037736	0.028302	0	0	0	8	1.000000	0.526697
LF_titlecase_P	17	0.146226	0.146226	0.080189	0	0	21	10	0.322581	0.535167
LF_titlecase_D	18	0.212264	0.212264	0.169811	0	0	32	13	0.288889	0.412569
LF_which	19	0.014151	0.014151	0.009434	0	0	0	3	1.000000	0.524031
LF_film	20	0.009434	0.009434	0.000000	0	0	0	2	1.000000	0.622605
LF_work	21	0.018868	0.018868	0.000000	0	0	0	4	1.000000	0.533173
LF_based	22	0.023585	0.023585	0.023585	0	0	3	2	0.400000	0.525683
LF_quote_or_year	23	0.301887	0.301887	0.240566	45	19	0	0	0.703125	0.782424
LF_movie_direct_P	24	0.099057	0.099057	0.042453	0	0	11	10	0.476190	0.555423
LF_genre_D	25	0.000000	0.000000	0.000000	0	0	0	0	NaN	0.535235
LF_genre_P	26	0.221698	0.221698	0.146226	0	0	17	30	0.638298	0.560012
LF_drama	27	0.028302	0.028302	0.028302	0	0	2	4	0.666667	0.533452
LF_nominat	28	0.056604	0.056604	0.037736	0	0	1	11	0.916667	0.536760
LF_final	29	0.014151	0.014151	0.014151	0	0	0	3	1.000000	0.522338
LF_reunit	30	0.018868	0.018868	0.018868	0	0	2	2	0.500000	0.537835
LF_document	31	0.018868	0.018868	0.009434	0	0	1	3	0.750000	0.617223
LF_first_capitalized_D	32	0.009434	0.009434	0.000000	0	0	2	0	0.000000	0.529287
LF_first_capitalized_P	33	0.466981	0.452830	0.367925	0	0	65	34	0.343434	0.500893
LF_names_between	34	0.367925	0.363208	0.330189	0	0	52	26	0.333333	0.755467
LF_partners	35	0.037736	0.037736	0.028302	0	0	0	8	1.000000	0.556621
LF_movie_between	36	0.047170	0.047170	0.047170	0	0	4	6	0.600000	0.517085
LF_actors_P	37	0.014151	0.014151	0.004717	0	0	1	2	0.666667	0.525041
LF_relations	38	0.070755	0.070755	0.070755	0	0	2	13	0.866667	0.539685
LF_distant_supervision	39	0.051887	0.047170	0.042453	11	0	0	0	1.000000	0.525226

3.4 Report the distribution of the training marginals

In [46]:

```
train_marginals = gen_model.marginals(L_train)
plt.hist(train_marginals, bins=20)
```

```
plt.show()
```



3.5 Explain (in your notebook) about your marginal distribution (max 3 sentences). Is it good or bad? Explain briefly.

The distribution seems good, given that most classifiers are very close to either 0 or 1, as while it was initially very bad, but as I kept adding more labeling functions based on the FPs and FNs the distribution began to improve. The current distribution differentiates well between the classes, although its clear from the plot that defining what is a match is much easier than defining what is NOT a match.

4. Training an Discriminative Model

In this final task, we'll use the noisy training labels we generated to train our end extraction model. In particular, we will be training a Bi-LSTM.

In [47]:

```
train_cands = session.query(performance_with_director).filter(performance_with_director.
split == 0).order_by(performance_with_director.id).all()
dev_cands   = session.query(performance_with_director).filter(performance_with_director.
split == 1).order_by(performance_with_director.id).all()
```

In [48]:

```
from snorkel.annotations import load_gold_labels

L_gold_dev  = load_gold_labels(session, annotator_name='gold', split=1)
```

Try tuning the hyper-parameters below to get your best F1 score

In [60]:

```
# ** STUDENT CODE

# TODO: TUNE YOUR HYPERPARAMETERS TO OBTAIN BEST RESULTS. WE EXPECT A F1-SCORE THAT IS HI
GHER THAN 0.7

from snorkel.learning.pytorch import LSTM

train_kwargs = {
    'lr':            0.01, # learning rate of the model
    'embedding_dim': 50,   # size of the feature vector
    'hidden_dim':    20,   # number of nodes in each layer in the model
    'n_epochs':      10,   # number of training epochs
    'dropout':       0.5,  # dropout rate (during learning)
    'batch_size':    32,   # training batch size
    'seed':          1701
}

lstm = LSTM(n_threads=-1)
lstm.train(train_cands, train_marginals, X_dev=dev_cands, Y_dev=L_gold_dev, **train_kwar
```

```
gs)
```

```
[LSTM] Training model
[LSTM] n_train=82  #epochs=10  batch size=32
[LSTM] Epoch 1 (0.39s) Average loss=0.675067 Dev F1=14.37
[LSTM] Epoch 2 (1.39s) Average loss=0.570447 Dev F1=80.39
[LSTM] Epoch 3 (2.31s) Average loss=0.483079 Dev F1=81.98
[LSTM] Epoch 4 (3.20s) Average loss=0.438753 Dev F1=80.90
[LSTM] Epoch 5 (4.08s) Average loss=0.437775 Dev F1=81.89
[LSTM] Epoch 6 (4.98s) Average loss=0.434732 Dev F1=81.89
[LSTM] Epoch 7 (5.93s) Average loss=0.405896 Dev F1=82.29
[LSTM] Epoch 8 (6.81s) Average loss=0.444036 Dev F1=81.61
[LSTM] Epoch 9 (7.72s) Average loss=0.400482 Dev F1=81.27
[LSTM] Model saved as <LSTM>
[LSTM] Epoch 10 (8.64s) Average loss=0.395008 Dev F1=81.98
[LSTM] Model saved as <LSTM>
[LSTM] Training done (9.20s)
[LSTM] Loaded model <LSTM>
```

Tune the hyper-parameters to get your best F1 score

Surprisingly, I found that 50 neurons in the hidden layer were actually too much for this ultra-small (82 samples) dataset, and when I cut the neurons to 20 I saw a good increase in F1 score (mostly because with more neurons recall starts to worsen).

I reduced the learning rate by an order of magnitudef which improved performance too. Presumably because given such a small, overfit-prone dataset, we really have to go slow then training.

Given the small train dataset size, one of the most significative changes I made was reducing the batch sizea from 64 to 32, which seems to have made overfitting slightly less of an issue.

Lastly, I've increased the Dropout rate from 0.2 to 0.33 as that showed improvements (further increases didn't).

Report performance of your final extractor

In [61]:

```
p, r, f1 = lstm.score(dev_cands, L_gold_dev)
print("Prec: {0:.3f}, Recall: {1:.3f}, F1 Score: {2:.3f}".format(p, r, f1))
```

```
Prec: 0.716, Recall: 0.959, F1 Score: 0.820
```

It took a lot of tweaking, but in the end I managed to convert the good distribution of marginals into good F1 Scores with the trained model. This was mostly a results of tuning hyperparameters with the small dataset in mind.

In [62]:

```
# TODO: MAKE SURE THE ABOVE CELL OUTPUT IS SHOWN IN YOUR PDF VERSION. THIS WILL BE YOUR A
NSWER FOR TASK 4
```

In [63]:

```
tp, fp, tn, fn = lstm.error_analysis(session, dev_cands, L_gold_dev)
```

```
========================================
Scores (Un-adjusted)
========================================
Pos. class accuracy: 0.959
Neg. class accuracy: 0.138
Precision            0.716
Recall               0.959
F1                   0.82
----------------------------------------
TP: 141 | FP: 56 | TN: 9 | FN: 6
========================================
```

Generally speaking my model's weak point is in accurately predicting non-matches, as defining what is *NOT* a match in a string proved to be the hardest aspect of developing the labeling functions, as can be seen by the difficulty of getting a bar stacked at 0 on the marginals.

The low accuracy for the negative class is a result of too many false positieves, which I was forced to content which precisely because of the difficulty of defining a non-match. If I use more relaxed labelings for that, then the model predicts nothing in the negative class, which makes for worse results, and thus I opted for a model with errs on the side of predicitng negative, which was the least of the evils.

In [64]:

```
# TODO: MAKE SURE THE ABOVE CELL OUTPUT IS SHOWN IN YOUR PDF VERSION. THIS WILL BE YOUR A
NSWER FOR TASK 4
```

Use your new model to extract relation in testing documents, and save it to JSON files.

In [65]:

```
# ** STUDENT CODE

# TODO: EXPORT YOUR PREDICTION OF THE DEV SET TO A CSV FILE

list_performances = []
list_directors = []
list_ids = []
for i in range(len(dev_cands)):
    list_ids.append(dev_cands[i][0].get_stable_id().split('::')[0])
    list_performances.append(dev_cands[i][0].get_attrib_span('words'))
    list_directors.append(dev_cands[i][1].get_attrib_span('words'))


dev_preds = lstm.predictions(dev_cands)

predictions_df = pd.DataFrame(data = {'id': list_ids,
                                      'performance': list_performances,
                                      'director': list_directors,
                                      'prediction': dev_preds})
print(f'predictions_df.shape: {predictions_df.shape}')
predictions_df.to_csv(HW_DIR / "Matheus_Schmitz_hw05_pred.dev.csv", index=False, header=
False)
```

```
predictions_df.shape: (212, 4)
```

Matheus Schmitz
USC ID: 5039286453