History, or Story?

Differentiating fact from fiction with machine learning.

Executive Summary

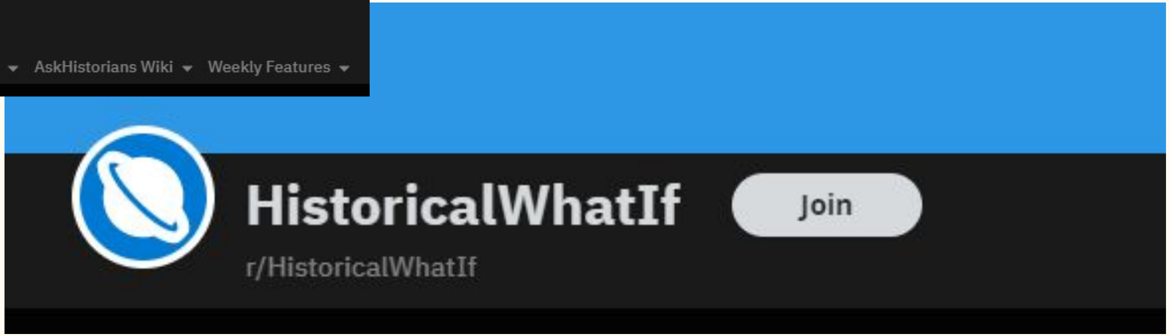
(Not complete)

Methodology

1. Identify a difficult classification, formulate problem statement to fit.
    a. “When talking about real history and alternate or speculative history, can we train a machine to recognize the difference with any accuracy, via NLP?”
2. Gather data representative of the difficulty inherent in the problem statement.
    a. Gather a list of subreddits related to history and peruse the content to inform our decision.
3. Apply diverse modeling tools to compare results.
    a. Contrasting methods is important here, so I selected KNN and SVC initially, as an Alpha-Omega split of low to high sophistication.

Problem Statement

Can we distinguish historical fact and scholarship from speculation and fiction?



Reddit — r/AskHistorians & r/HistoricalWhatIf

- Two related subreddits were selected from a shortlist of popular History subs.
- r/History, r/HistoryNetwork, and r/AlternateHistory were also considered but ultimately dismissed because the two subs selected were the best match for overall sample size.

Model Selection

KNN - Simple, Black Box, brute force method. Surprisingly effective.

Decision Tree - More transparent, explicable. Difficult to prune correctly.

Random Forest - DT fit on a variety of sub-samples of the dataset.

SVC - Supervised learning, very robust, creates a hyperplane for classification.

Known to be an excellent model for text classification.

Data Scraping from Reddit using PushShift API

Goal = 25,000 datapoints from each.

Actual = 20k from AskHistorians, 16k from WhatIf

Post-Cleaning = 15k and 9k.

In retrospect I would have de-duplicated earlier to keep balance. However, my datasets were not so unbalanced (~61/39) as to cause real problems.

```python
def psquery(sub, kind = 'submission', interval = 7, q = 5, skip = 0):
    subfields = ['title', 'selftext', 'subreddit', 'created_utc', 'author', 'num_comments', 'score', 'i

    # establish base url and stem
    roots = f"https://api.pushshift.io/reddit/search/{kind}" # also known as the "API endpoint"
    trunk = f"{roots}?subreddit={sub}&size=100" # always pulling max of 500

    # instantiate empty list for temp storage
    posts = []

    # implement for loop with `time.sleep(2)`
    for i in range(1, q + 1):
        url = "{}&after={}d".format(trunk, ((interval * i) + skip))
        print("Querying from: " + url)
        response = requests.get(url)
        assert response.status_code == 200
        harvest = response.json()['data']
        df = pd.DataFrame.from_dict(harvest)
        posts.append(df)
        time.sleep(2)
```

```python
# pd.concat storage list
full = pd.concat(posts, sort=False)

# if submission
if kind == "submission":
    # select desired columns
    full = full[subfields]
    # drop duplicates
    full.drop_duplicates(inplace = True)
    # select `is_self` == True
    full = full.loc[full['is_self'] == True]

# create `timestamp` column
full['timestamp'] = full["created_utc"].map(dt.date.fromtimestamp)

print("Query Complete!")
return full
```

Workflow

1. Collect datasets
2. Clean, combine, and process data
    a. Remove duplicates - cut my sample size in half.
    b. Standard cleaning - lowercase, remove punctuation,
    c. Count Vectorizer - included bi & trigrams
    d. Lemmatize / Stem - not done
3. Apply models
4. Evaluate models

Baseline Accuracy

With 61.5% of my dataset falling into the 0 category (AskHistorians), a semi-educated guess has any given example about 150% more likely to be real than a what-if.

Any model with an accuracy score lower than 61.5% is failing to converge.

```
# Baseline accuracy
y_test.value_counts(normalize=True)
```

executed in 29ms, finished 13:14:06 2021-01-29

```
0    0.615879
1    0.384121
Name: whatif, dtype: float64
```

k-Nearest Neighbors

KNN:	Training set:	0.8026146592370339
	Test set:	0.6925961641487196

Not great, but at least it's outperforming the null model.

Less features was definitely more in this case.

Decision Trees

DTs:	Training set:	0.6159451350192885
	Test set:	0.6158791385406621

(Failed to converge)

Random Forest

Random Forest:	Training set:	0.9924989284183455
	Test set:	0.8271723990142505

Best so far - interesting that it so significantly outperformed DTs when it is just a meta-classifier on top of the same core method.

Support Vector Classifier (SVM)

Random Forest:	Training set:	0.9344906415202172
	Test set:	0.8357441337190614

Model Evaluation

- All models need additional tuning.
- Best results with RF by far.
- For the sake of explicability, I hope to use insights gained from the success of RF to better tune and prune DTs.

Conclusions and Recommendations