I investigate different approaches to predicting the popularity of a news article, given article metadata and content. In a 2015 CS229 project, He Ren and Quan Yang used a random forest to achieve 69% accuracy in classifying articles as either popular (>1400 shares) or not. I aim to break the baseline by engineering new features on the scraped article content itself, combining what I’ve learned in CS229 and various NLP techniques. I obtained the dataset of article urls and metadata from UCI’s Online News Popularity Dataset. To create a robust classifier, I used beautifulsoup to scrape all the articles of its author and content.

**Models Rationale**

**Ridge Regression** - Regression, particularly Linear, tries to find a model that maximizes the correlation between features and popularity. Support Vector Classification - This tries to find a margin that maximally separates popular and non-popular articles. There are metadata features, like sentiment and polarity, that may help construct an effective separation boundary.

**Kernels** - The cosine kernel will work well on the word2vec features, but it’s a question what kernel to use for the LSA features. A custom “composite” kernel may get the best of both worlds.

**Random Forests** - This is based off of the baseline set by the previous project with 500 trees; with the new features I engineer, it’s a question whether the same sampling/splitting strategy will work.

**Neural Networks** - It’s not an ML project without this. If logistic regression performs well, it’s natural to add layers.

**Motivation & Data**

I investigate different approaches to predicting the popularity of a news article, given article metadata and content. In a 2015 CS229 project, He Ren and Quan Yang used a random forest to achieve 69% accuracy in classifying articles as either popular (>1400 shares) or not. I aim to break the baseline by engineering new features on the scraped article content itself, combining what I’ve learned in CS229 and various NLP techniques. I obtained the dataset of article urls and metadata from UCI’s Online News Popularity Dataset. To create a robust classifier, I used beautifulsoup to scrape all the articles of its author and content.

**Metadata Features**

**Attributes and Statistics**

- Title: Article popularity appears to be negligible long-term trend. I hypothesize the popularity gained over time "cancels out" with the site’s long-term rise in visitors.
- Date: Seven features are formed via the one-hot encoding for day of the week of publication. This may be useful for splitting a decision tree or a separation boundary.
- Channel: Eight features are formed via the one-hot encoding for channel of the article - similar to date.

**Summary Statistics**

- Tokens: These include number of tokens in title and content, number of unique tokens, etc. They may be weakly correlated and mildly useful.
- Number of _ (images, videos, references, etc.): These may be weakly correlated as well.
- References: (min/max shares of referenced article): This is useful, but adds a lot of variance. This once again explains why random forests worked so well.

**Processed Metrics**

- Sentiment, Polarity, Subjectivity - These are metrics in their own right. They are continuous and very useful for regression.
- Best/Worst/Average Keyword - This may explain why random forests worked so well.
- LDA - There are features that model commonalities to modeled topics; these numbers are useful after splitting into random forests.

**Conclusions (Being Finalized)**

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 Score</th>
<th>Average Recall</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ridge Regression</td>
<td>0.71</td>
<td>0.65</td>
<td>This is useful as a baseline, and it closely performs well after optimizing the few thousand features extracted from the content.</td>
</tr>
<tr>
<td>Support Vector Classification</td>
<td>0.65</td>
<td>0.65</td>
<td>This doesn't perform as well, and understandably so, as features are less correlated. The outputs are highly connected.</td>
</tr>
<tr>
<td>Random Forests</td>
<td>0.68</td>
<td>0.71</td>
<td>This is the baseline set by the CS229 project that helped inspire mine.</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>0.70</td>
<td>0.69</td>
<td>This is more of a black box. Three layers with logistic activations perform well. There's a lot of hyperparameter tuning I have yet to experiment with, so take this with a grain of salt.</td>
</tr>
</tbody>
</table>

**Future**

Ultimately, the goal is to dig deeper into the article extract the best predictors of popularity, but first I am curious how much predictive power is in the url and first sentence of the article. Afterwards, I want to investigate a more recurrent approach by ordering the features (i.e. url, keywords -> title -> first sentence -> first paragraph -> etc.) and having the algorithm halt once it has “enough information” (this is similar to how humans read articles). Via rewards, the agent learns to prioritize which features are more important and which to look at first.

**Models Rationale**

**Ridge Regression** - Regression, particularly Linear, tries to find a model that maximizes the correlation between features and popularity. Support Vector Classification - This tries to find a margin that maximally separates popular and non-popular articles. There are metadata features, like sentiment and polarity, that may help construct an effective separation boundary.

**Kernels** - The cosine kernel will work well on the word2vec features, but it’s a question what kernel to use for the LSA features. A custom “composite” kernel may get the best of both worlds.

**Random Forests** - This is based off of the baseline set by the previous project with 500 trees; with the new features I engineer, it’s a question whether the same sampling/splitting strategy will work.

**Neural Networks** - It’s not an ML project without this. If logistic regression performs well, it’s natural to add layers.

**Motivation & Data**

I investigate different approaches to predicting the popularity of a news article, given article metadata and content. In a 2015 CS229 project, He Ren and Quan Yang used a random forest to achieve 69% accuracy in classifying articles as either popular (>1400 shares) or not. I aim to break the baseline by engineering new features on the scraped article content itself, combining what I’ve learned in CS229 and various NLP techniques. I obtained the dataset of article urls and metadata from UCI’s Online News Popularity Dataset. To create a robust classifier, I used beautifulsoup to scrape all the articles of its author and content.