Tuning efforts:

HMM Tagger

The Hidden Markov Model (HMM) tagger involves several tuning efforts focused on different parameters to optimize the tagger's performance.

Base HMM test accuracy without any tuning:
“test_ood_hmm.txt”: ***0.23110079575596817***
“test_hmm.txt”: ***0.24208079046788725***

The first parameter tunerd was to change the number of ***hidden states***. The variation in the number of states helps determine the optimal complexity of the model, balancing between overfitting and underfitting.

The range of hidden states considered is from 2 to 9 (num_states_range = range(2, 10)), which correspond to different parts of speech or tags that the model can recognize. This improved the accuracy by 3 times

New test accuracy:
“test_ood_hmm.txt”: ***0.7370689655172413***
“test_hmm.txt”: ***0.7948270851496658***

Next, was to use ***estimation techniques***. Three different estimators are used: Laplace (laplace_estimator), Expected Likelihood Estimation (ele_estimator), and Witten-Bell (witten_bell_estimator). These estimators influence how the model estimates probabilities, particularly in dealing with unseen events or sparse data

- LaplaceProbDist adds a small count to all probabilities, thereby smoothing them.
- ELEProbDist uses expected likelihood for smoothing
- WittenBellProbDist is another smoothing technique

Alongside there was also an effort to use 5-fold ***cross-validation*** (num_folds = 5) for HMM to evaluate the model's performance and ensure that the assessment is robust and less prone to biases due to particular data splits. However, this lead to similar results and took longer time, and hence was removed

All in all, the code uses a method of ***best model selection***, i.e. iterates through all combinations of the number of states and estimators. For each combination, the model's accuracy is assessed using the accuracy method

The combination yielding the highest accuracy is chosen as the best model configuration. The selected best model is then used to tag the test dataset, and its accuracy is computed by comparing the predicted tags against the actual tags

Final Model Evaluation

For HMM, the best number of hidden states was found to be ***2*** and, the best estimator was the ***witten bell estimator*** with the best accuracy

Final test accuracy:
“test_ood_hmm.txt”: ***0.8189655172413793***
“test_hmm.txt”: ***0.8622493461203139***

Brill Tagger

For brill tagger we first started with just a ***Unigram tagger***, with a ***backoff*** to a general 'NN' tag. Additionally, during training the max rules were set to 10.

Initial test accuracy scores for test.txt:
test_ood_brill: ***0.80252***
test_brill: ***0.82476***

We believed that we could further enhance this score and set out searching for methods to do so. First we changed out the unigram tagger for a ***more complex trigram tagger***, whose backoff architecture would be

Trigram -> Bigram -> Unigram -> 'NN'

This improved our accuracy too (test.txt)
test_ood_brill.txt: ***0.804568792397933***
test_brill.txt: ***0.8262133100842778***

After, we decided to delve into what our brill tagger was unable to tag correctly.
We found quite a few words that were being tagger NN by default due to our final backoff tagger. To correct this we implemented a **RegexpTagger** instead and created rules that we believed were generalized enough to increase our accuracy on the data set.

This improved our accuracy on out test sets too:
test_ood_brill.txt: ***0.8545678477887489***
test_brill.txt: ***0.8718395815170009***

While going through the trace of our tagger during training, we found that we have been under-utilizing our rules. We ***changed max rules*** to many different numbers such as 50, 100, 1000; and realized that after 100, the accuracy does not improve by much. This improved our accuracy too

Throughout this tuning process, we did not feel the need to make changes to the template as the default worked well enough.

Final Model Evaluation

To conclude, the best parameter set for our Brill Tagger consists of a ***trigram-based tagging strategy*** with an extensive set of regex-based rules through a ***RegexpTagger***, and an increased number of transformation rules (***up to 100***).

Final test accuracy:
test_ood_brill.txt: ***0.8624005305039788***
test_brill: ***0.8750363266492299***

Tagger performance:

Tagger	Input	Output	Tagger Performance
hmm	data/test.txt	output/test_hmm.txt	0.8622493461203139
hmm	data/test_ood.txt	output/test_ood_hmm.txt	0.8189655172413793
brill	data/test.txt	output/test_brill.txt	0.8747457134553909
brill	data/test_ood.txt	output/test_ood_brill.txt	0.8624005305039788

Tagger Errors:

The most (>10) misclassifications for **HMM** was done for the followings tags

test.txt
JJ misclassified most with NN - 26
NN misclassified most with JJ - 22
NNS misclassified most with NN - 58
NNP misclassified most with NN - 42

test_ood.txt
VBD misclassified most with VBP - 45
JJ misclassified most with NN - 26
NN misclassified most with JJ - 21
NNS misclassified most with NN - 53
VBG misclassified most with NN - 11
VBN misclassified most with NN - 12

The most (>10) misclassifications for **BRILL** was done for the followings tags

test.txt
VB misclassified most with NN - 43
JJ misclassified most with NN - 29
TO misclassified most with IN - 14
NN misclassified most with JJ - 12
VBN misclassified most with VBD - 25
RB misclassified most with NN - 13
NNP misclassified most with NN - 77

test_ood.txt
VBD misclassified most with VBN - 24
JJ misclassified most with NN - 33
VBN misclassified most with VBD - 34
NN misclassified most with JJ - 12
DT misclassified most with IN - 13
PRP misclassified most with NN - 15
VB misclassified most with NN - 15
NNP misclassified most with NN - 31
UH misclassified most with NN - 12

We first generated a table that would give a breakdown of where our tagger was failing, so that we could more accurately understand its pitfalls. We also created a most incorrect tags file to see the most pairs of correct-incorrect occurrences.

HMM

- Adjectives (JJ) like ‘synonymous’ in “and synonymous” may have been misclassified as nouns (NN) because they can function as nouns in certain contexts “is synonymous”. This contextual flexibility can confuse the tagger.
- Nouns (NN) such as ‘marble’ or ‘red’ could be mistaken for adjectives (JJ) due to their use as modifiers in noun phrases (like "marble floor" or "red car").
- Plural nouns (NNS) often share a root with their singular counterparts (NN). If the training data didn't adequately represent certain plural forms, the tagger might default to the more commonly seen singular form (“towers”, “fixtures”).
- Proper nouns (NNP) were misclassified as common nouns (NN) since they are uncommon and thus not represented in the training data (“Tyler”).
- Past tense verbs (VBD) may be confused with non-3rd person singular present verbs (VBP) if the training data lacks examples of these verbs in past tense contexts, especially if they have similar forms in both tenses (like 'watched' in "I watched" vs. "I watched a movie").
- Gerunds (VBG) can be particularly challenging as they are verbs functioning as nouns (NN) (e.g., closing in "The store is closing"). Without clear contextual indicators, these might be mistaken for common nouns.

BRILL

- Words like 'Drizzle' that were at the beginning of the sentence were misclassified NN, due to the word being unseen in the training dataset. Most of the sentences in training didn't start with a VB. Words like 'think' were classified as VBP instead of VB due to the capitalization of the first letter.
- Words like 'braised' were classified as VBD due to the inability of our RegexpTagger to differentiate between them.
- Many words like 'sorry' were classified as NN due to it being unseen during training, and that our RegexpTagger was not fit to be able to identify such words as JJ.
- Many words like 'Phoenix' we classified as NN. This is due to how ambiguous the line between NNP and NN is, and due to the fact that almost all of these words were unseen during training.

Comparison of tagger errors

Commonalities:

- Both taggers faced challenges with unseen words during training.
- Both had misclassifications between JJ and NN, indicating inherent difficulties in distinguishing between adjectives and nouns.
    - Adjectives like 'synonymous' or 'braised' being misclassified as nouns by both taggers.
- The misclassifications for in domain were less varied in comparison to out of domain, for both taggers.
- Both HMM and Brill taggers frequently misclassified proper nouns (NNP) as common nouns (NN). This issue often arises due to the uncommon nature of some proper nouns, their absence in the training set, or failure to recognize capitalization cues.

Differences:

- HMM misclassifications involved more variety in part-of-speech tags (e.g., VBD, VBP, RB, RP).
    - This is because Brill tags everything it doesn't understand as NN. HMMs, being probabilistic models, rely heavily on the frequency and patterns of tags in the training data. If past tense verbs (VBD) are underrepresented or presented in different contexts in the training data compared to OOD text, the model may struggle to correctly identify them.
- Brill Tagger had specific challenges with VB and capitalization affecting VB vs. VBP classifications.
- Brill Tagger utilized RegexpTagger to address specific misclassifications, while HMM focused on different estimation techniques.
- HMMs consider a limited context (typically the immediate previous tag) for making predictions. This limited window might not capture the broader context needed to distinguish between VBD and VBP in OOD text, where contextual cues might be more nuanced or spread out across a sentence. Brill was able to overcome this.