1. Data Preprocessing

(no written deliverable required)

2. Most Frequent Tag Baseline

(a) (no written deliverable required)

(b) F_1 score on dev set: 0.80

Full output reproduced here for completeness:

```
Token-level confusion matrix:
go\gu   PER      ORG      LOC      MISC    O
PER     3033.00  20.00    29.00    5.00    62.00
ORG     494.00   1139.00  157.00   43.00   259.00
LOC     287.00   146.00   1510.00  21.00   130.00
MISC    219.00   29.00    52.00    842.00  126.00
O       290.00   46.00    6.00     27.00   42390.00

Token-level scores:
label acc  prec rec  f1
PER   0.97 0.70 0.96 0.81
ORG   0.98 0.83 0.54 0.66
LOC   0.98 0.86 0.72 0.78
MISC  0.99 0.90 0.66 0.76
O     0.98 0.99 0.99 0.99
micro 0.98 0.95 0.95 0.95
macro 0.98 0.85 0.78 0.80
not-O 0.98 0.78 0.76 0.77

Entity level P/R/F1: 0.78/0.82/0.80
```

3. HMM Tagger

(a) (no written deliverable required)

(b) Our pruning policy is to drop out a pair of tags with lowest value in each layer. Layers smaller than 4 are not pruned. The pruning function implementation in pseudo-code:

```
def prune(self):
if len(self.layer_dict) > MIN_LAYER_SIZE_TO_PRUNE:
    weak_key = key with lowest value
    del self.layer_dict[weak_key]
return
```

(c) F_1 on dev set: 0.84

Full output of hmm.py:

```
Token-level confusion matrix:
go\gu    PER      ORG      LOC      MISC     O
PER      3036.00  56.00    19.00    12.00    26.00
ORG      327.00   1467.00  125.00   41.00    132.00
LOC      271.00   227.00   1542.00  25.00    29.00
MISC     180.00   58.00    19.00    930.00   81.00
O        268.00   234.00   15.00    76.00    42166.00

Token-level scores:
label acc   prec  rec   f1
PER   0.98  0.74  0.96  0.84
ORG   0.98  0.72  0.70  0.71
LOC   0.99  0.90  0.74  0.81
MISC  0.99  0.86  0.73  0.79
O     0.98  0.99  0.99  0.99
micro 0.98  0.96  0.96  0.96
macro 0.98  0.84  0.82  0.83
not-O 0.98  0.78  0.81  0.80

Entity level P/R/F1: 0.81/0.86/0.84
```

(d) `P(x_8 = 'LOC') = 0.024332529500131664`

(e) Let A, B, X, Y be hidden states (plus start state *) and t, x, y be possible tokens. Suppose the training corpus consists of the sentences:

- t/A x/X (sentence repeated 891 times in training corpus)
- t/A y/Y (sentence repeated 9 times in training corpus)
- t/B y/Y (sentence repeated 100 times in training corpus)

Where each q/R indicates token q with labelled hidden state R.

By ML estimation, the transition probabilities in Q are

- $q_{*A} = P(A|*) = 0.9$
- $q_{*B} = P(B|*) = 0.1$
- $q_{AX} = P(X|A) = 0.99$
- $q_{AY} = P(Y|A) = 0.01$
- $q_{BX} = P(X|B) = 0$
- $q_{BY} = P(Y|B) = 1$
- (all other transition probabilities are zero)

and the emission probabilities in E are

- $e_{At} = e_{Bt} = e_{Xx} = e_{Yy} = 1$

– (all other emissison probabilities are zero)

Now consider the sentence $s = ty$.

The greedy decoding algorithm first calculates the possibilities for the first hidden state:

– $q_{*A}e_{At} = 0.9 \cdot 1 = 0.9 \leftarrow$ winner

– $q_{*B}e_{Bt} = 0.1 \cdot 1 = 0.1$

– $q_{*X}e_{Xt} = 0 \cdot 0 = 0$

– $q_{*Y}e_{Yt} = 0 \cdot 0 = 0$

so it sets the first hidden state to A. Then it calculates the possibilities for the second hidden state:

– $q_{AA}e_{Ay} = 0 \cdot 0 = 0$

– $q_{AB}e_{By} = 0 \cdot 0 = 0$

– $q_{AX}e_{Xy} = 0.99 \cdot 0 = 0$

– $q_{AY}e_{Yy} = 0.01 \cdot 1 = 0.01 \leftarrow$ winner

Therefore the output of the greedy decoding algorithm is AY, which has total probability $q_{*A}e_{At}q_{AY}e_{Yy} = 0.9 \cdot 0.01 = 0.009$.

On the other hand, we calculate that the total probability of the hidden state sequence BY is $q_{*B}e_{Bt}q_{BY}e_{Yy} = 0.1 \cdot 1 \cdot 1 \cdot 1 = 0.1$, which is higher.

Therefore Viterbi will return a different result than greedy decoding.

4. MEMM Tagger

(a) (no written deliverable required)

(b) (no written deliverable required)

(c) Optimizations used:

– At each step, the predict(...) function of LogisticRegression is called on all candidates at once (one feature matrix where each row is a candidate path). This is more efficient than calling it on each candidate path separately.

– Scores and candidate paths are cached in the variable best_paths

– For numerical stability, summed log-probabilities are used instead of the product of probabilities.

– Paths (partially tagged sentences) that are not extended on a step are pruned (overwritten).

(d) F_1 on dev set: 0.85 for both greedy and Viterbi decoding.

Greedy decoding output:

```
    Token-level confusion matrix:
go\gu   PER      ORG      LOC      MISC     O
PER     2778.00  53.00    71.00    22.00    225.00
ORG     108.00   1607.00  80.00    44.00    253.00
LOC     41.00    101.00   1729.00  27.00    196.00
MISC    40.00    38.00    25.00    1004.00  161.00
```

```
O        55.00    87.00    42.00    64.00    42511.00

Token-level scores:
label acc   prec  rec   f1
PER   0.99  0.92  0.88  0.90
ORG   0.99  0.85  0.77  0.81
LOC   0.99  0.89  0.83  0.86
MISC  0.99  0.86  0.79  0.83
O     0.98  0.98  0.99  0.99
micro 0.99  0.97  0.97  0.97
macro 0.99  0.90  0.85  0.88
not-O 0.99  0.89  0.83  0.86

Entity level P/R/F1: 0.87/0.82/0.85
```

Viterbi decoding output:

```
        Token-level confusion matrix:
go\gu    PER      ORG      LOC      MISC     O
PER      2779.00  54.00    69.00    22.00    225.00
ORG      109.00   1615.00  78.00    43.00    247.00
LOC      41.00    100.00   1730.00  27.00    196.00
MISC     39.00    36.00    25.00    1009.00  159.00
O        53.00    90.00    42.00    63.00    42511.00

Token-level scores:
label acc   prec  rec   f1
PER   0.99  0.92  0.88  0.90
ORG   0.99  0.85  0.77  0.81
LOC   0.99  0.89  0.83  0.86
MISC  0.99  0.87  0.80  0.83
O     0.98  0.98  0.99  0.99
micro 0.99  0.97  0.97  0.97
macro 0.99  0.90  0.85  0.88
not-O 0.99  0.89  0.83  0.86

Entity level P/R/F1: 0.87/0.83/0.85
```

(e) From the confusion matrix we see the largest source of error is false negatives (entities being falsely tagged as O). Some examples from the corpus:

- " Seven percent lowerCase lowerCase problems and a lowerCase lowerCase for money from lowerCase , " said initCap initCap , chief technical analyst at Raymond James .
  "Raymond" was tagged as O instead of PER.
- FTSE close $ containsDigitAndPeriod $ containsDigitAndPeriod
  "FTSE" was tagged as O instead of MISC.

- The girl was abducted from her school in initCap near Berlin on Monday

  "initCap" (preprocessed rare word) was tagged as O instead of LOC.

Among confusion between entity types, some of the most common are PER being mistakenly tagged as ORG and ORG being mistakenly tagged as LOC. Examples:

- Traders said the initCap 's decision to lowerCase a lowerCase lowerCase at the July allCaps meeting has cast more focus on every piece of U.S. economic news .

  First "initCap" (preprocessed rare word) was tagged as LOC instead of ORG.
- initCap signed a pact with Moscow in April to create a strong economic and political union which he believes could grow into a federation .

  First "initCap" (preprocessed rare word) was tagged as ORG instead of PER.

Both of these are understandable error types. False negatives mean that the model may be too conservative in its predictions, not predicting an entity because it cannot reach sufficient confidence. The confusions between PER/ORG/LOC seem especially common for "initCap" tokens which makes sense because when the identity of the word is lost, just knowing that it starts with a capital letter does not give much information about whether it is a person, organization, or place, and this is not always clear from context.

5. BiLSTM Tagger

(a) Without masking, some of the terms in the loss function would be $CE(y^{(t)}, \hat{y}^{(t)})$ where t is the timestep of padding. In other words, the loss will be higher as long as the model does not emit "0-vector" output in padding positions, and the gradient will be biased towards changes that make the model give such output in padding positions. We expect that this will make the model take longer to learn NER tagging in non-padded positions. Masking solves this because all of these terms will have added coefficient 0 and thus will not contribute to the loss.

(b) (no written deliverable required)

(c) See q5/predictions.conll and q5/log for deliverables. Best dev F_1 score: 0.89