Task

- ☑ Take any approach
- ☑ Explore more features
- ☑ Explain Improvements

Java files can be found in my hw3 file.

SimpleNERecognizer - Baseline Highest Score 8th iteration: 93.63	HerreraNERecognizer – With additional features Highest Score 37th iteration: 97.44

From HerreraNERecognizer.java

Approach: **Subgradient Perceptron** | Iterations: **50** | Alpha: **0.00001**

For starters I ended up using the Subgradient Perceptron approach with 50 iterations along with an alpha of 0.00001. I tried AllVsOne but I kept getting worse results. This sort of make sense as the data is huge and updating all the hyper planes with each token seemed over cumbersome and complicated. What didn't make sense to me is why the score never changed from 83.32 for OnevsAll despite whatever I tried. Increasing the number of iterations for subgradient perceptron definitely helped. This made sense as I gave my perceptron time and attempts to adjust hyper planes to accommodate all my features to maximize accuracy. As for the alpha of 0.0001, this came about midway through testing as I found lowering the constant improved accuracy to a degree. After becoming too small, alpha would reduce accuracy.

From HerreraFeatureExtractor - Features Explored

Previous: **Labels** & **Words** & **Suffix** | Forward: **Words** | Current: **Word** & **Prefixes** & **Suffix**

I can categorize my features into three main types. Previous looking, forward looking and current looking. Let's take a look at each of them briefly.

Previous Labels – 2 features

Here I made 2 features getting index -1, index -2 labels. These turned out to be helpful at predicating the current label of the index token. This makes sense as a sequence of labels could help predict the next label. However when including index-3 label, it hurt the accuracy of the program.

```
Example:  if (index-1 >= 0)
                features.add(new StringFeature("f6", tokens.get(index-1).getLabel()));
```

Previous Words – 2 features

Similar in nature to the previous. Knowing the words of index-1 and index-2 helped the accuracy of decoding. I forgot to mention for this one and previous labels, that the if-statements are to avoid null pointer exceptions.

```
Example:  if (index-2 >= 0)
                features.add(new StringFeature("f5", tokens.get(index-2).getWord()));
```

Previous Suffix

This Helped a bit as knowing the previous suffix gives a clue to the next part of speech or label. I tried previous prefix and going more back to like index-2 but that didn't help accuracy.

```
Example:  if (index-1 >= 0)  {      int Tlengthim1 = tokens.get(index-1).getWord().length();
            for(int w =1; w<=3; w++)      {
              if (w <= Tlengthim1)
                features.add(new StringFeature("f10", tokens.get(index-1).getWord().substring(Tlengthim1 -w, Tlengthim1)));
                                    }                  }
```

Forward Words – 2 features

Using forward words helped in decoding the current word. This makes sense as it gives context to the current word similar to how using previous words do. I used 2 words forward because when using 3 or more, the results were less accurate than just using 2.

Example:
```
if (index+1 < tokens.size()-1)
        features.add(new StringFeature("f1", tokens.get(index+1).getWord()));
```

Current Word

This was part of the base code given.

Current word's Suffix

This was a huge boost to accuracy. This makes sense as an ending like "tion" or "ing" or "ed" or "s" does help predict the part of speech of a given word. I played with the length of the prefix and concluded 5 was the best. While 5 doesn't make as much sense as 4 for lettered endings, up'ing the number to 5 helped in decoding accuracy.

Example:
```
for(int j =1; j<=5; j++)   {
   if (j <= Tlength)
     features.add(new StringFeature("f9", tokens.get(index).getWord().substring(Tlength -j, Tlength)));   }
```

Current word's Prefix

Similar to the current words suffix, this helps in determining the part of speech greatly. Similarly as before, I played with the length of the prefix to conclude 3 was the best given the training data.

Example:
```
for(int i =1; i<=5; i++)     {
   if (i <= Tlength)
     features.add(new StringFeature("f8", tokens.get(index).getWord().substring(0, i)));   }
```

Closing Thoughts

Compared to the Simple Recognizer and SimpleFeatureExtracter that we were given, my Herrera version does a better job at decoding. My HerreraFeatureExtractor looks both backward on a word to word basis and on a label to label basis. On top of that it looks a bit forward as it takes into account the next two words and the coming prefix. Given all that I've said, my program's main improvements and strengths over the class default comes from the quality of features I have.

With further Tinkering I yielded a higher score of 97.59 by adding these two additional features. Rational is in line with forward words and suffix's and prefix's in general.

```
//forward word Prefix
if (index+1 < tokens.size()-1)
{int Tlength2 =tokens.get(index+1).getWord().length();

for(int i =1; i<=3; i++)
{   if (i <= Tlength2)
{
    features.add(new StringFeature("f11", tokens.get(index+1).getWord().substring(0, i)));
}}}

//forward word suffix
if (index+1 < tokens.size()-1)
{int Tlength2 =tokens.get(index+1).getWord().length();

for(int j =1; j<=5; j++)
{
    if (j <= Tlength2)
        features.add(new StringFeature("f12", tokens.get(index+1).getWord().substring(Tlengt
}
}
```

```
58: 97.30
59: 97.50
60: 97.36
61: 97.48
62: 97.43
63: 97.59
```