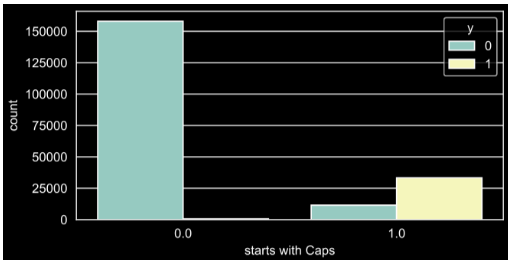
NLP assignment 1

Ram Cohen & Jonathan fuchs

Features we tried:

1. **Word with Capital first letter** – makes sense for entities (Jerusalem, Yossi, Germany…)

```
lambda sentence: [word[0].isupper() for word in sentence[0]]
```

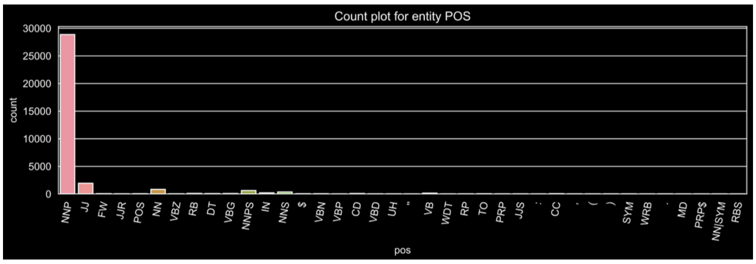


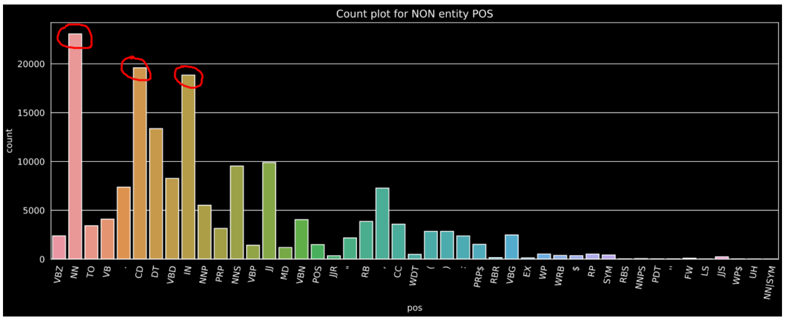
2. **Word location in the sentence** – usually the first is with capital first letter – so together with (1) should help distinguish them.

```
lambda sentence: [ii for ii in range(len(sentence[0]))],
```

3. **Part of speech (pos) –** we tried to find one or more a pos that is more frequent with entities or with non-entities.

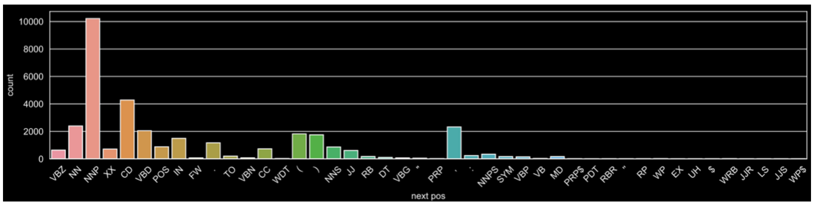
```
lambda sentence: [True if pos == 'NNP' else False for pos in sentence[1]]
lambda sentence: [True if pos in ('NN', 'CD', 'IN', 'DT') else False for pos in sentence[1]]
```





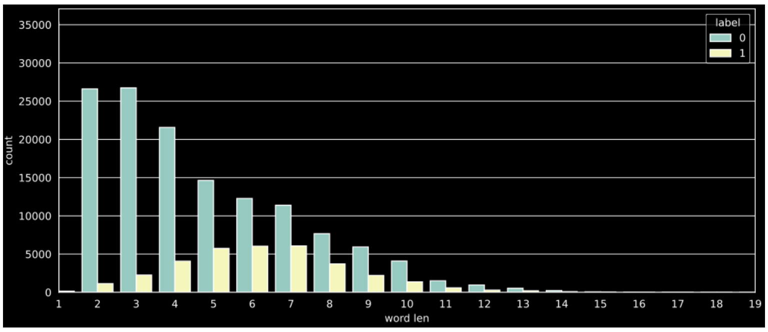
4. **Last pos** – like with (3) only looking at the previous word
5. **Next pos** - like with (3) only looking at the next word

```
lambda sentence: [True if last_pos == 'NNP' else False for last_pos in sentence[2]],
lambda sentence: [True if next_pos == 'NNP' else False for next_pos in sentence[3]],
```



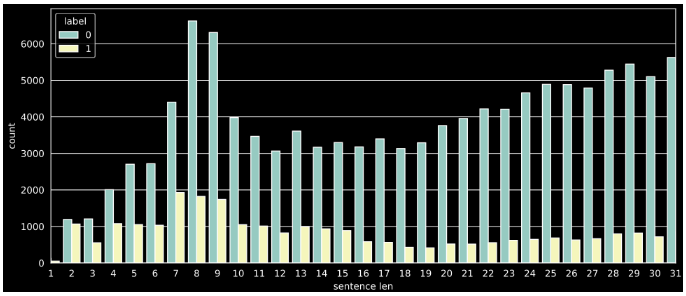
6. **Word length –** we tried looking at the word length, and is there a difference between entities or non-entities

```
lambda sentence: [len(word) > 3 for word in sentence[0]]
```



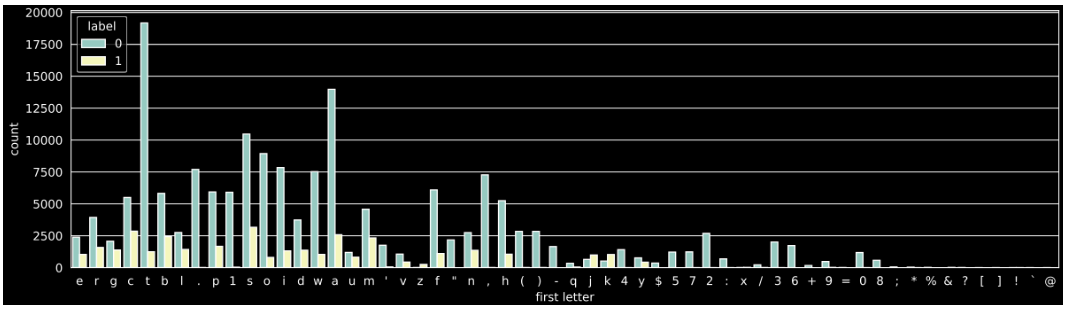
7. **Sentence length** – like with (6), but looking at the sentence length instead, **not very promising**

```
lambda sentence: [len(sentence[0]) > 15 for word in sentence[0]]
```



8. **First letter (lower case)** – entities and non-entities have different first letters, **not very promising**

```
lambda sentence: [word[0].lower() in 't,f.o' for word in sentence[0]],
lambda sentence: [word[0].lower() in 'jkyzv' for word in sentence[0]]
```



Model and hyper parameters:

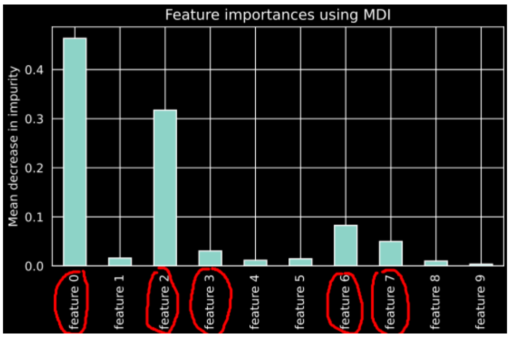
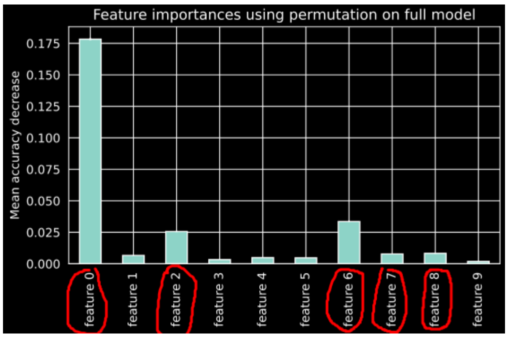
Models we tried (with all available features), (random_state=42):

- GradientBoostingClassifier(max_depth=15): 96.59% on eval
- AdaBoostClassifier(n_estimators=100): 95.49% on eval
- KneighborsClassifier(n_neighbors=7): 96.10% on eval
- SVC(kernel='rbf'): 95.7% on eval
- <u>RandomForestClassifier(n_estimators=300, max_depth=13)</u>: **97.02%**

Choosing Random forest, as it is fast and good for tabular data, a simple hyperparameter optimization search have been conducted (estimators and depth) and the above parameters were chosen.

Selecting top 5 features:

The selection was based on feature importance, some features were combined to reduce the number of features while keeping some of their effect.

1. mean decrease in impurity:	2. Permutation importance:
	

Final 5 features:

1. Feature 0: Capital letter:

```
lambda sentence: [word[0].isupper() for word in sentence[0]
```

2. Feature 2: Part of speech – for entities:

```
lambda sentence: [True if pos in ('NNP', 'NNPS', 'JJS') else False for pos in sentence[1]]
```

3. Feature 3: Part of speech – for non entities:

```
lambda sentence: [True if pos in ('NN', 'CD', 'IN', 'DT') else False for pos in sentence[1]]
```

4. Feature 6: word length:

```
lambda sentence: [len(word) for word in sentence[0]]
```

5. Feature 7: sentence length:

```
lambda sentence: [len(sentence[0]) for word in sentence[0]]
```

Confusion matrix (on eval data-set):

Accuracy: 96.06%, Precision: 84.96% , recall: 92.93%, f1: 88.77%

The model is better at classifying non-entities and a little biased to finding entities at the expense of its accuracy.