

# README file for Assignment 1

Jiangshan Luo  
U89971259

## 1. Code File

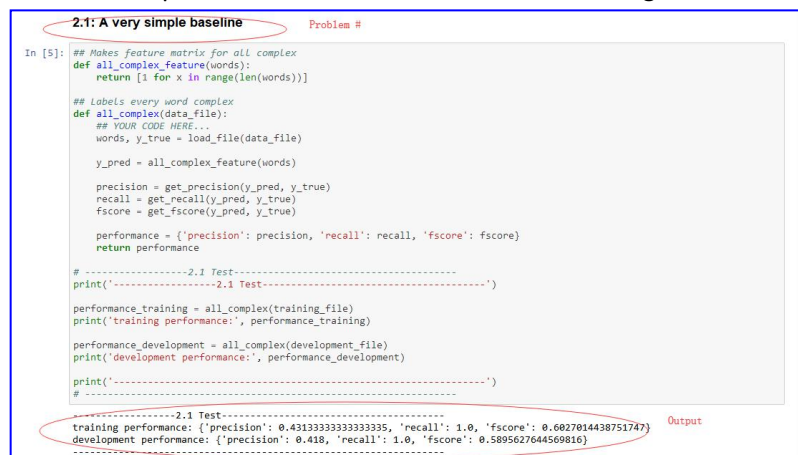
The code file contains workable code and data that it needs.

There are **3 different types of the code** (Jupyter Notebook, python source code, HTML). Please pick whichever you can run. Note that the outputs can be easily visualized in Jupyter Notebook and HTML, so it might be better to view those two.

There is also a **data file** (train, development, test data), **two neural network HDF5 models**, **syllables.py** and **ngram\_counts.txt.gz**.

## 2. Source Code Clarification

The question **titles** are marked about each code block, and the **results** of them are right below. And also, the order of the questions are the same as which in the assignment document.



The screenshot shows a Jupyter Notebook interface. At the top, a title '2.1: A very simple baseline' is circled in red. Below it, a code cell contains Python code for a word classifier. The code defines a function `all_complex_feature` that takes a list of words and returns a feature matrix. It then defines a function `all_complex` that takes a data file path, loads the data, and returns a performance dictionary. The code cell is followed by a '2.1 Test' section that prints the training and development performance. The output of the test is shown at the bottom, with the training performance dictionary circled in red. The output is as follows:

```
In [5]: ## Makes feature matrix for all complex
def all_complex_feature(words):
    return [1 for x in range(len(words))]

## Labels every word complex
def all_complex(data_file):
    ## YOUR CODE HERE...
    words, y_true = load_file(data_file)
    y_pred = all_complex_feature(words)

    precision = get_precision(y_pred, y_true)
    recall = get_recall(y_pred, y_true)
    fscore = get_fscore(y_pred, y_true)

    performance = {'precision': precision, 'recall': recall, 'fscore': fscore}
    return performance

# -----2.1 Test-----
print('-----2.1 Test-----')

performance_training = all_complex(training_file)
print('training performance:', performance_training)

performance_development = all_complex(development_file)
print('development performance:', performance_development)

print('-----2.1 Test-----')
# -----2.1 Test-----
training performance: {'precision': 0.4313333333333333, 'recall': 1.0, 'fscore': 0.6027014438751747}
development performance: {'precision': 0.418, 'recall': 1.0, 'fscore': 0.5895627644569816}
```

## 3. Clarification for my complex word classifier

In terms of features, I applied four features in my model. They are **word length**, **word frequency**, **syllable number**, and **synonym number**. (they are either used previously in this assignment or mentioned in the papers provided below the assignment document)

For the model, there are several sub-models that I used to classify the labels of words. Namely, **Naive Bayes**, **Support Vector Machine**, **K Nearest Neighbors** and a **Feed-Forward Neural Network**. After gathering outputs from these models, there is a **final decision maker model** (another FNN that is trained to make right decisions given the predictions of the four models) that decide which class should the word belong.

The reason I use a decision maker model is to compensate for the drawbacks of each model. For example, the NB model can figure out most of the complex words (high recall) but also misclassify the complex words (low precision), whereas SVM has higher precision but lower recall.