# Final Project AI Fall 2019

## DeftEval-2020 Task 6:

## Extract textual definitions from naturally occurring text

## Presented by:

| | |
|---|---|
| • *Ahmed Mohamed Hamdi* | *4671* |
| • *Fagr Ahmed Refaat* | *4814* |
| • *Hania Mohamed Hani* | *4697* |

# 1 Introduction

It is gaining popularity as a prior step for constructing taxonomies, ontologies, automatic glossaries or dictionary entries. These fields of application motivate greater interest in well-formed encyclopedic text from which to extract definitions, and therefore DE for academic or lay discourse has received less attention. Historically, Definition Extraction has been a popular topic in NLP research, but has been limited to well defined, structured, and narrow conditions. For which we're using the following classifiers:

1- **Naive Bayes:** calculates the probabilities for every factor. Then it selects the outcome with highest probability. This classifier assumes the features are independent. Hence, the word naive.

2- **Logistic Regression:** Logistic regression is a statistical method for analyzing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

3- **K-Nearest Neighbor (KNN):** K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.

4- **Support Vector Machine (SVM):** is a discriminating classifier formally defined by a separating hyper-plane.

5- **Decision tree:** In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

6- **Random Forest:** A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

## 2 *Approach*

The Training Data set is a collection of enriched labeled data to help us train our models, the more data we have, the sharper the model accuracy

- The data was in the form of columns, [TOKEN] [SOURCE] [START CHAR] [END CHAR] [TAG] [TAG ID] [ROOT ID] [RELATION].
- We used a script to convert it from the sequence/relation labeling format to classification format. This produces files in the following tab-delineated format: [SENTENCE] [HAS DEF]. This is intended for Subtask 1: Sentence Classification.

## 3 *Data Pre-Processing and Feature Extraction*

After converting, we have two separate folders, "Testing" and "Training". Each folder consists of data in text format. We extract the data from each file and put into 2 separate arrays "string" and "labels", labels are in the form of 1 or 0.

- The first thing we did was opening the .deft file and we took the text column content and put it in an array, and the labels column which has the labels of each feature and put them in a separate array.
- Cleaning the data was done by converting all data to lower case, removing any symbol or punctuation, removing numbers and stem data using NLTK built-in function stem to reduce all similar words to their origin to improve accuracy.
- **Stemming and Lemmatization** are Text Normalization (or sometimes called Word Normalization) techniques in the field of Natural Language Processing that are used to prepare text, words, and documents for further processing by returning the world to its source. LancasterStemmer is simple, but heavy stemming due to iterations and over-stemming may occur. Over-stemming causes the stems to be not linguistic or they may have no meaning.
- **Tf-idf** stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.
- We used the TFIDF vectorizer, so we had to import the first library we used, Sci-kit Learn, this has a built-in function stop_words = English to remove the unnecessary redundant English words from each entry in the text list.

## 4 *Vectorization and Word Embedding*

Text data requires special preparation before you can start using it for predictive modeling.

The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).

- Although a short process, it's an extremely important step because all classifiers work on vectors not actual strings. We then use the fit transform function that takes the list of texts and transforms it to a vector representing the words that correspond in a particular list with the entire corpus, and the transform function to build matrix with same vocabulary from the training data neglecting any new vocabulary in testing data. Now our training data is ready for the classification phase.
- Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

**-TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

**-IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$$\begin{aligned} tf_{i,j} &= \text{number of occurrences of } i \text{ in } j \\ df_i &= \text{number of documents containing } i \\ N &= \text{total number of documents} \end{aligned}$$

## *5 Classifiers*

1. **Naive Bayes** is a simple, yet effective and commonly-used, machine learning classifier. It can be represented using a very simple Bayesian network. Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection. Seeing how famous Naive Bayes has been used in text classification we decided to start with it in our implementation.

Using the same Sci-kit library we imported MultinomialNB which is suitable for classification with discrete features. We use the fit function in this classifier taking as parameters the list of training texts and the training labels, both of which are vectors of integer value. After this step we use the function predict on a test data set file and we print the results of the predictions returned in an array and the result classifies each sentence according to the fitted data from the training set.

2. **Logistic Regression** Logistic Regression is a go-to method for binary classification (which is what were going for); this algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. We need the output of the algorithm to be class variable, i.e. 0 for no, 1 for yes. Therefore, we are squashing the output of the linear equation into a range of [0,1]. To squash the predicted value between 0 and 1, we use the sigmoid function. Its widely used for text classification and is known for exhibiting pretty accurate results, also its very simple to implement.

We import the Logistic Regression library again from the Scit-kit library and we use the exact functions used previously in Naive Bayes, fit() and predict() taking in the training data and the test data respectively and resulting in predictions in the form of an array.

3. **K-Nearest Neighbor (KNN)** The KNN classifier is based on the assumption that the classification of an instance is most similar to the classification of other instances that are nearby in the vector space. Compared to other text categorization methods such as Bayesian classifier, KNN does not rely on prior probabilities, and it is computationally efficient.

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry, this algorithm is one of the simplest classification algorithms it also has a low calculation time.

The challenge is to find the perfect k to run our algorithm with trying to avoid under fitting or over fitting. KNN algorithm is used to classify by finding the K nearest matches in training data and then using the label of closest matches to predict.

We import the KNeighborsClassifier from the Sci-kit library and we use the same functions previously explained in the other classifiers with only the difference of setting the k to the desired value before starting the fitting operation, the result as the other classifiers is an array

4. **Support Vector Machine (SVM)** Support vector machines is an algorithm that determines the best decision boundary between vectors that belong to a given group (or category) and vectors that do not belong to it. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

SVMs can produce accurate, unique and robust classification results on a sound theoretical basis, even when input data are non-monotone and non-linearly separable. So they can help to evaluate more relevant information in a convenient way. Since they linearize data on an implicit basis by means of kernel transformation, the accuracy of results does not rely on the quality of human expertise judgment for the optimal choice of the linearization function of non-linear input data.

Svm library is imported from Sci-kit library and we use the function SVC(support vector classifier) and we set the kernel parameter to linear. And we fit the data and predict them as mentioned before in the other classifiers.

5. **Decision Tree** Decision trees implicitly perform variable screening or feature selection. It can handle both numerical and categorical data, also the effort required for Data preparation is minimal.

We imported DecisionTreeClassifier from the Sci-kit library to use for fitting our model and predicting labels for the test data. We chose gini as our criterion because it's better and faster than entropy at calculating the gain that decides where the tree splits.

6. **Random Forest** Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. They are adaptable at solving any kind of problem at hand. Random Forest is a versatile machine learning method capable of performing both regression and classification tasks.

It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model. One of benefits of Random forest is the power of handle large data set with higher dimensionality.

It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods. We imported the RandomForestClassifier from Sci-kit ensemble library and we use it to fit and predict our model.

# 6 Testing Data

➢ ***Confusion Matrix***

| | | Actual Label | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted Label** | Positive | **True Positive (TP)** | **False Positive (FP)** |
| | Negative | **False Negative (FN)** | **True Negative (TN)** |

- True Positives (TP) - These are the correctly predicted positive values which mean that the value of actual class is yes and the value of predicted class is also yes.
- True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.
- False Positives (FP) – When actual class is no and predicted class is yes.
- False Negatives (FN) – When actual class is yes but predicted class in no.

➢ ***Accuracy*** - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations

Accuracy = TP+TN/TP+FP+FN+TN

➢ ***Precision*** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

➢ ***Recall*** (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class.

Recall = TP/TP+FN

➢ ***F1 score*** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution.

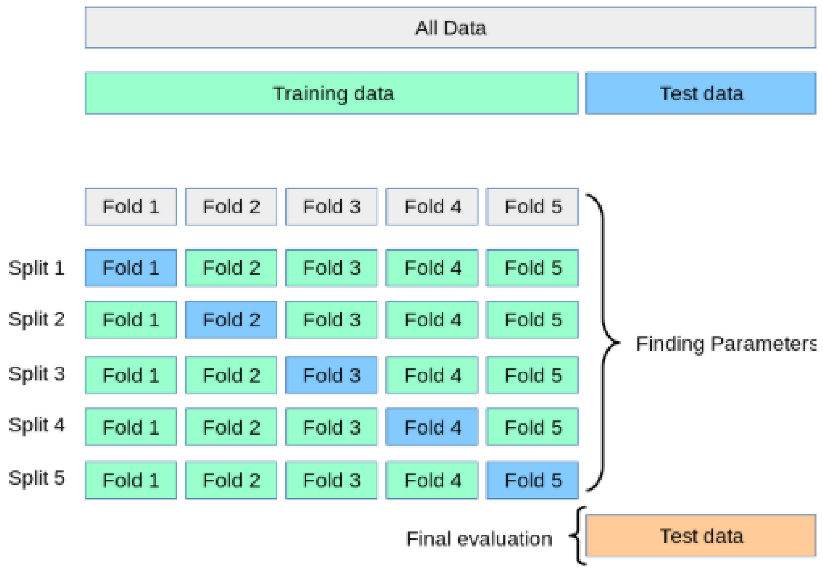
F1 Score = 2*(Recall * Precision) / (Recall + Precision)

# 7 Validation

Cross-Validation is an essential tool in the Data Science. It allows us to utilize our data better.

When we're building a machine learning model using some data, we often split our data into training and validation/test sets. The training set is used to train the model, and the validation/test set is used to validate it on data it has never seen before. The classic approach is to do a simple 80%-20% split, sometimes with different values like 70%-30% or 90%-10%. In cross-validation, we do more than one split. We can do 3, 5, 10 or any K number of splits. Those splits called Folds, and there are many strategies we can create these folds with.

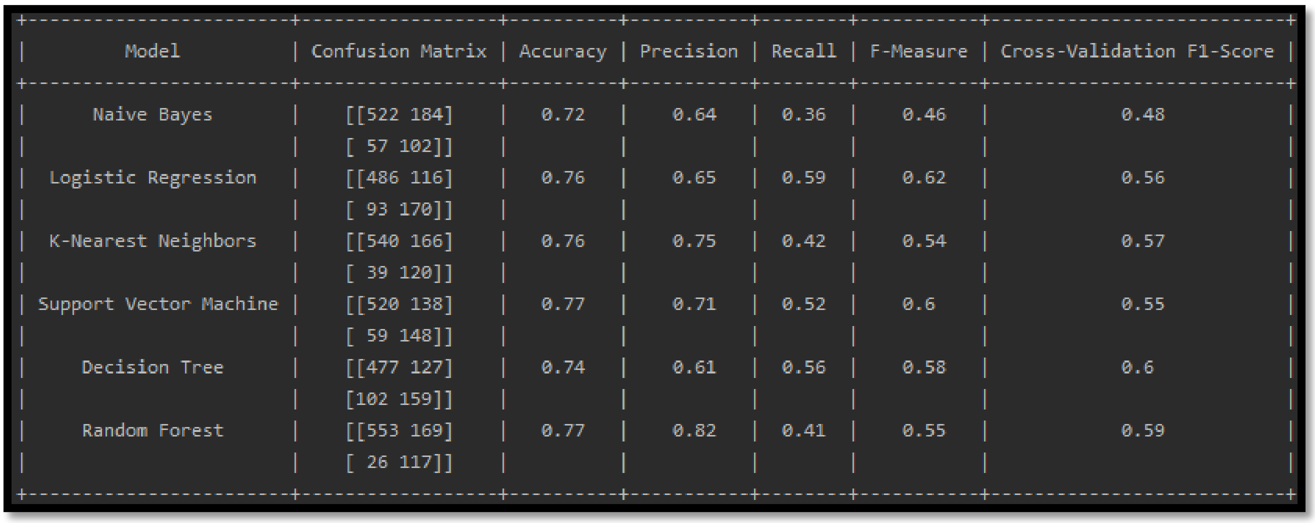
We used the Simple K-Folds — We split our data into K parts, we used K=5. If we have 18157 instances in our dataset, We split it into five parts. We then build five different models, each model is trained on four parts and tested on the fifth. Our first model is trained on part 1, 2, 3, 4 and tested on part 5. Our second model is trained to on part 1, 3, 4, 5 and tested on part 2 and so on. We used a function called cross_val_score() imported from sklearn which takes as inputs the training set and the fit model and give us a mean of f1 score over k-fold cross validated 5 consecutive times with different splits each time.



The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the re-sampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller. To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

## 8 *Results*

We ran our code on a test file full of thousands of sentences, we used each classifier on the file and we calculated the metric measures of all of them and the results were as following:



```
+------------------------+-----------------+----------+-----------+--------+-----------+--------------------------+
|         Model          | Confusion Matrix | Accuracy | Precision | Recall | F-Measure | Cross-Validation F1-Score |
+------------------------+-----------------+----------+-----------+--------+-----------+--------------------------+
|      Naive Bayes       |    [[522 184]    |   0.72   |    0.64   |  0.36  |    0.46   |           0.48           |
|                        |    [ 57 102]]    |          |           |        |           |                          |
|  Logistic Regression   |    [[486 116]    |   0.76   |    0.65   |  0.59  |    0.62   |           0.56           |
|                        |    [ 93 170]]    |          |           |        |           |                          |
|  K-Nearest Neighbors   |    [[540 166]    |   0.76   |    0.75   |  0.42  |    0.54   |           0.57           |
|                        |    [ 39 120]]    |          |           |        |           |                          |
| Support Vector Machine |    [[520 138]    |   0.77   |    0.71   |  0.52  |    0.6    |           0.55           |
|                        |    [ 59 148]]    |          |           |        |           |                          |
|     Decision Tree      |    [[477 127]    |   0.74   |    0.61   |  0.56  |    0.58   |           0.6            |
|                        |    [102 159]]    |          |           |        |           |                          |
|     Random Forest      |    [[553 169]    |   0.77   |    0.82   |  0.41  |    0.55   |           0.59           |
|                        |    [ 26 117]]    |          |           |        |           |                          |
+------------------------+-----------------+----------+-----------+--------+-----------+--------------------------+
```

## 9 *Tuning parameters*

We needed to improve accuracy of the results from our classifiers and one way to do that is tuning the parameters of each classifier in a way that maximizes the performance of classification. Some of the parameters we changed from their default state were:

1. **Naive Bayes** There is 3 parameters: alpha, fit prior, class prior. Alpha: a smoothing parameter (int) to increase the probabilities in our non-maximum likelihood equation by 1 to make everything non-zero. Fit prior: whether to learn class prior probabilities or not.

2. **Logistic Regression** penalty parameter used to regularize the model. It basically adds the penalty as model complexity increases to avoid over fitting. We have used L2 which is called Ridge Regression. With testing we set the c parameter to 10 which has provided the best accuracy and increased the number of iterations to 1000 after which the model converges

3. **K-Nearest Neighbor (KNN)** was tuned using a code implemented to find the best K to run the algorithm with.

4. **Support Vector Machine (SVM)** we tuned some important parameters kernel and C. we set the kernel to: linear because a large number of features is more likely that the data is linearly separable, C: Penalty parameter C of the error term, it controls the tradeoff between smooth decision boundary and classifying the training points correctly setting it to 1 proved the best results.

## 10 Statistics

- Number of Train Data Sentences: 18157
    - Definition: 6014
    - Not Definition: 12143
- Number of Test Data Sentences: 865
    - Definition: 286
    - Not Definition: 579
- Training Vocabulary before pre-processing: 27234
- Testing Vocabulary before pre-processing: 4728
- Training Vocabulary after pre-processing: 11360
- Testing Vocabulary after pre-processing:  2738

## 11 Conclusion

In our implementation what we gathered is that the best classifier was the SVM and it worked with accuracy 77% this is due to the fact that the SVM works really well with clear margin of separation and it's heavily recommended for text classification projects.

## 12 References

- http://www.tfidf.com/
- https://machinelearningmastery.com/k-fold-cross-validation/
- https://scikit-learn.org/stable/modules/cross_validation.html
- https://scikit-learn.org/stable/modules/model_evaluation.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html#sklearn.metrics.classification_report
- https://machinelearningmastery.com/clean-text-machine-learning-python/
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f