

# Course of Web Information Retrieval

## Homework 3

Simone Agostinelli 1523559, Giacomo Vettraino 1594722

### 1 Ham/Spam Classifier

#### 1.1 KNN Classifier

##### 1.1.1 Dataset

Our dataset is composed by 347 text files representing ham and spam comments related to YouTube videos. These files are composed according to the following table:

	Training Set	Test Set
Spam	122	53
Ham	120	52

##### 1.1.2 Vectorizer and Classifier Parameters

Firstly we consider the process of vectorization that we have to apply in order to transform our text files into a matrix of Tfidf features. For this step we have:

**tokenizer**  $\in \{None, stemming\_tokenizer, stemming\_tokenizer\_stopwords\_filter\}$   
override the string tokenization function if its value is different from *None*;

**ngram\_range**  $\in \{(1, 1), (1, 2), (1, 3)\}$   
specify the lower and upper boundary of the range of n-values for different n-grams to be extracted;

While, for the classification phase we have:

**n\_neighbors**  $\in \{1, 3, 5, 7, 9\}$

specify the number of neighbors to use for classifying points;

**weights**  $\in \{ "uniform", "distance" \}$

specify the weight function in prediction ;

### 1.1.3 Training-Validation Phase

Training and validation are performed by using the sklearn Python function **GridSearchCV** in an automated fashion. Indeed, it performs an exhaustive search of the best parameters values configuration by fitting the specific model (or a series of models transformation if pipeline is used); furthermore, it can output the best estimator and the best parameters configuration by computing scores on this combinations using a scoring function chosen by the user.

In particular, in our case, firstly we decided to use **Pipeline** sklearn function, in order to construct the different steps that we want to cross-validate: vectorization and classifier fitting. **Pipeline** function has input parameter:

**steps**

List of (name, transform) tuples that are chained, in the order in which they are chained, with the last object an estimator.

```
1 | vectorizer = TfidfVectorizer(strip_accents= None ,
2 | preprocessor = None ,)
3 | knn = KNeighborsClassifier()
4 | pipeline = Pipeline([('vect', vectorizer), ('knn', knn),])
```

Then, we can directly make use of **GridSearchCV** function, in order to setup the automated search for the best parameter configuration.

**GridSearchCV** has as input parameters:

**estimator**

This is the estimator on which we want to perform our grid search; in this case we set it to the pipeline just created, as its last step represent an estimator, our KNN Classifier;

**param\_grid**

Dictionary with parameters names (string) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored; this enables searching over any sequence of parameter settings;

**scoring**

A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`; in this case Matthews correlation coefficient is used;

**n\_jobs**

Number of jobs to run in parallel, in this way we can control the parallelism of our program, in order to speed up it running time;

**cv**

determines the cross-validation splitting strategy; an integer specify the number of folds in a KFold, 10 in our case;

```
1 | parameters = {
2 | 'vect__tokenizer': [None, stemming_tokenizer,
3 | stemming_tokenizer_stopwords_filter],
4 | 'vect__ngram_range': [(1, 1), (1, 2), (1,3)],
5 | 'knn__n_neighbors': [1,3,5,7,9],
6 | 'knn__weights': ["uniform", "distance"]
7 | }
8 | grid_search = GridSearchCV(
9 | pipeline,
10 | parameters,
11 | scoring = metrics.make_scorer(metrics.matthews_corrcoef),
12 | cv = 10,
13 | n_jobs = 4)
```

### 1.1.4 Best Parameters Values

After performing cross-validation, as explained before, we have found that the best parameters configuration is:

Parameter	Value
knn__n_neighbors	9
knn__n_weights	distance
vect__ngram_range	(1,2)
vect__tokenizer	None

### 1.1.5 Results for Classification

Output of `metrics.classification_report`:

1	-----
2	
3	precision recall f1-score support
4	Ham 0.83 0.96 0.89 52
5	Spam 0.96 0.81 0.88 53
6	
7	avg / total 0.90 0.89 0.89 105
8	-----

The **Confusion Matrix**:

	Predicted-Ham	Predicted-Spam
True-Ham	50	2
True-Spam	10	43

The **Normalized-Accuracy** value: 0.885714285714

The **Mattiews Correlation Coefficient**: 0.780832919631

## 2 Sentiment Analysis

### 2.1 KNN Classifier

#### 2.1.1 Dataset

Our dataset is composed by 1115 text files representing positive and negative sentences. These files are composed according to the following table:

	Training Set	Test Set
Negative	249	250
Positive	308	308

### 2.1.2 Vectorizer and Classifier Parameters

Firstly we consider the process of vectorization that we have to apply in order to transform our text files into a matrix of Tfidf features. For this step we have:

**tokenizer**  $\in \{None, stemming\_tokenizer, stemming\_tokenizer\_stopwords\_filter\}$   
 override the string tokenization function if its value is different from *None*;

**ngram\_range**  $\in \{(1, 1), (1, 2), (1, 3)\}$   
 specify the lower and upper boundary of the range of n-values for different n-grams to be extracted;

While, for the classification phase we have:

**n\_neighbors**  $\in \{1, 3, 5, 7, 9\}$   
 specify the number of neighbors to use for classifying points;

**weights**  $\in \{uniform, distance\}$   
 specify the weight function in prediction ;

### 2.1.3 Training-Validation Phase

#### 2.1.4 Best Parameters Values

After performing cross-validation, as explained before, we have found that the best parameters configuration is:

Parameter	Value
knn__n_neighbors	3
knn__n_weights	distance
vect__ngram_range	(1,2)
vect__tokenizer	stemming_tokenizer_stopwords_filter

### 2.1.5 Results for Classification

Output of `metrics.classification_report`:

1	-----				
2		precision	recall	f1-score	support
3					
4	Positive	0.85	0.94	0.89	308
5	negative	0.92	0.80	0.85	250
6					
7	avg / total	0.88	0.88	0.88	558
8	-----				

### The Confusion Matrix:

	Predicted-Ham	Predicted-Spam
True-Ham	290	18
True-Spam	51	199

The **Normalized-Accuracy** value: 0.876344086022

The Matthews Correlation Coefficient: 0.752375671874

## 2.2 MultinomialNB Classifier

### 2.2.1 Dataset

Our dataset is composed by 1115 text files representing positive and negative sentences. These files are composed according to the following table:

	Training Set	Test Set
Negative	249	250
Positive	308	308

### 2.2.2 Vectorizer and Classifier Parameters

Firstly we consider the process of vectorization that we have to apply in order to transform our text files into a matrix of Tfidf features. For this step we have:

**tokenizer**  $\in \{None, stemming\_tokenizer, stemming\_tokenizer\_stopwords\_filter\}$   
 override the string tokenization function if its value is different from *None*;

**ngram\_range**  $\in \{(1, 1), (1, 2), (1, 3)\}$   
 specify the lower and upper boundary of the range of n-values for different n-grams to be extracted;

While, for the classification phase we have:

**alpha**  $\in \{0.001, 0.01, 1, 10\}$  specify the additive smoothing parameter;

### 2.2.3 Parallelism

### 2.2.4 Best Parameters Values

After performing cross-validation, as explained before, we have found that the best parameters configuration is:

Parameter	Value
mnf__alpha	1
vect__ngram_range	(1,1)
vect__tokenizer	stemming_tokenizer_stopwords_filter

## 2.2.5 Results for Classification

Output of `metrics.classification_report`:

```
1 | -----
2 |               precision    recall  f1-score   support
3 |
4 |   Positive         0.90      0.98      0.94       308
5 |   negative         0.98      0.86      0.92       250
6 |
7 | avg / total         0.93      0.93      0.93       558
8 | -----
```

The **Confusion Matrix**:

	Predicted-Ham	Predicted-Spam
True-Ham	303	5
True-Spam	34	216

The **Normalized-Accuracy** value: 0.930107526882

The **Mattiews Correlation Coefficient**: 0.862006201146

## 2.3 SVC

### 2.3.1 Dataset

Our dataset is composed by 1115 text files representing positive and negative sentences. These files are composed according to the following table:

	Training Set	Test Set
Negative	249	250
Positive	308	308



### 2.3.2 Vectorizer and Classifier Parameters

Firstly we consider the process of vectorization that we have to apply in order to transform our text files into a matrix of Tfidf features. For this step we have:

**tokenizer**  $\in \{None, stemming\_tokenizer, stemming\_tokenizer\_stopwords\_filter\}$   
override the string tokenization function if its value is different from *None*;

**ngram\_range**  $\in \{(1, 1), (1, 2), (1, 3)\}$   
specify the lower and upper boundary of the range of n-values for different n-grams to be extracted;

While, for the classification phase we have:

**C**  $\in \{0.01, 0.1, 1.0, 10.0, 100.0\}$   
specify the penalty parameter C value of the error term ;

### 2.3.3 Parallelism

### 2.3.4 Best Parameters Values

After performing cross-validation, as explained before, we have found that the best parameters configuration is:

Parameter	Value
svc__C : 10	
vect__ngram_range	(1,3)
vect__tokenizer	stemming_tokenizer_stopwords_filter

### 2.3.5 Results for Classification

Output of **metrics.classification\_report**:

1	-----				
2		precision	recall	f1-score	support
3					
4	Positive	0.96	0.97	0.97	308
5	negative	0.97	0.95	0.96	250

6					
7	avg / total	0.96	0.96	0.96	558
8	-----				

The **Confusion Matrix**:

	Predicted-Ham	Predicted-Spam
True-Ham	300	8
True-Spam	13	237

The **Normalized-Accuracy** value: 0.962365591398

The **Mattiews Correlation Coefficient**: 0.923917742518