

Spam detection

Francesco Bellezza



Scopo

Cercare di costruire un modello che riesca a riconoscere quali sms sono spam:

<https://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html>

Software di utilizzo: Weka



Preprocessing

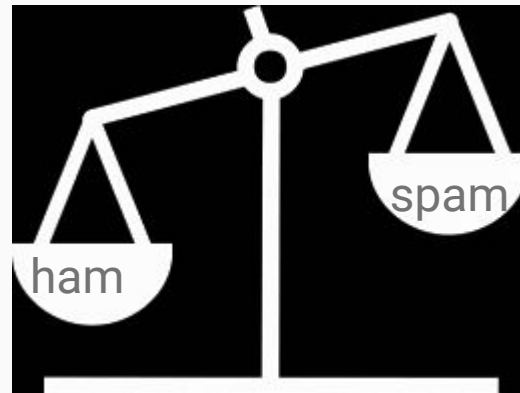
I dati non sono bilanciati all'interno delle classi:
quindi è necessario effettuare un bilanciamento!
Sono presenti più sms non spam.



Inoltre, bisognerà effettuare la trasformazione
dei vari sms in array di parole.



86%

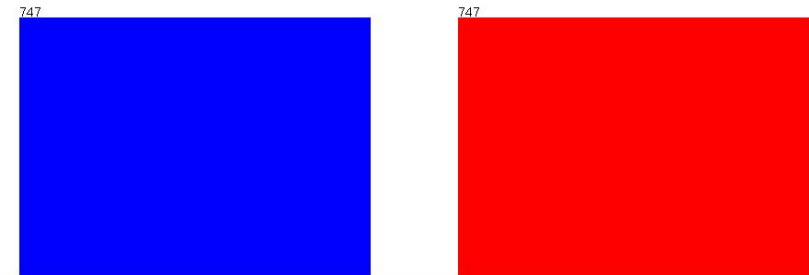
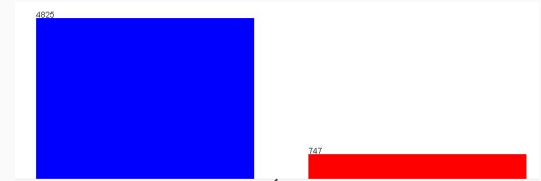
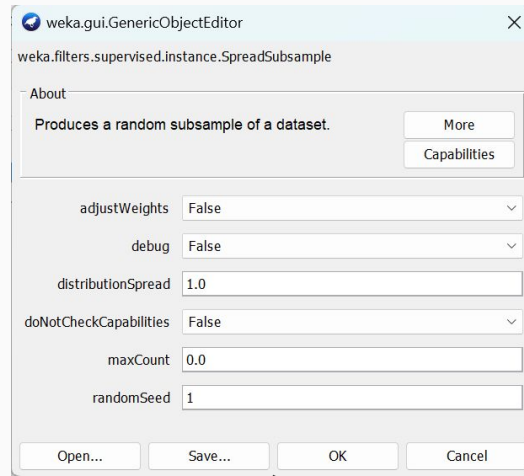


Rappresenta il numero di sms non spam che sono presenti nel dataset...

Rischio overfitting rispetto ai dati che abbiamo!

Subsample

Per evitare il problema di overfitting, decidiamo di generare un sottocampione in modo tale che le classi siano bilanciate.



StringToWordVector

Abbiamo trasformato i testi dei vari sms in bag of words.

weka.gui.GenericObjectEditor

weka.filters.unsupervised.attribute.StringToWordVector

Converts string attributes into a set of numeric attributes representing word occurrence information from the text contained in the strings.

More

Capabilities

IDFTTransform False

TFTTransform False

attributeIndices first-last

attributeNamePrefix

debug False

dictionaryFileToSaveTo

doNotCheckCapabilities False

doNotOperateOnPerClassBasis False

invertSelection False

lowerCaseTokens True

minTermFreq 1

normalizeDocLength No normalization

outputWordCounts True

periodicPruning -1.0

saveDictionaryInBinaryForm False

stemmer Choose **SnowballStemmer**

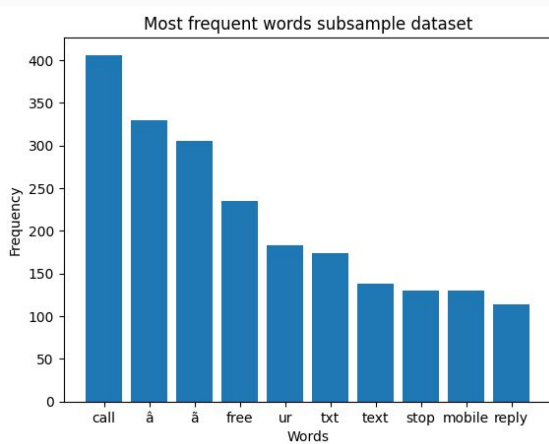
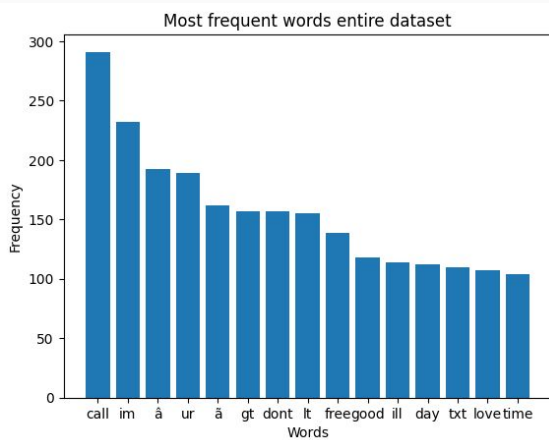
stopwordsHandler Choose **Rainbow**

tokenizer Choose **WordTokenizer -delimiters "**

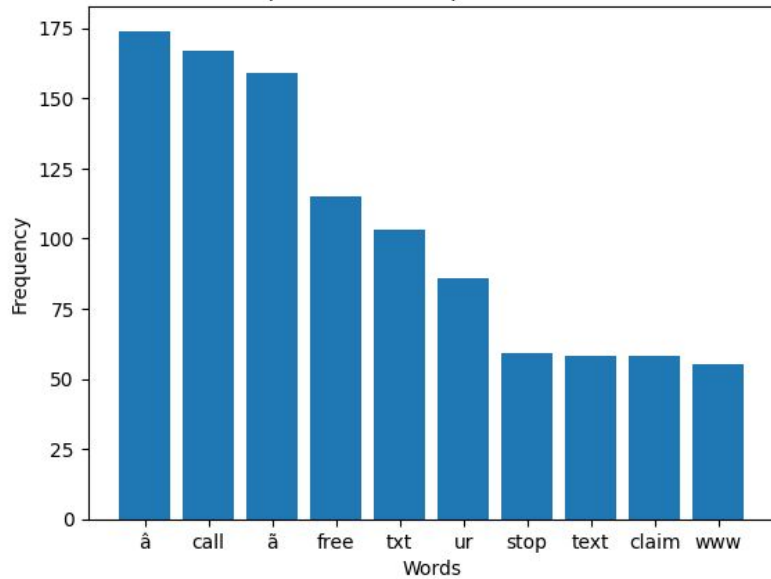
No.	1: Class Nominal	2: aah Numeric	3: aaniye Numeric	4: aathi Numeric	5: abel Numeric
1	ham	0.0	0.0	0.0	0.0
2	ham	0.0	0.0	0.0	0.0
3	ham	0.0	0.0	0.0	0.0
4	ham	0.0	0.0	0.0	0.0
5	ham	0.0	0.0	0.0	0.0
6	ham	0.0	0.0	0.0	0.0
7	ham	0.0	0.0	0.0	0.0
8	ham	0.0	0.0	0.0	0.0
9	ham	0.0	0.0	0.0	0.0
10	ham	0.0	0.0	0.0	0.0
11	ham	0.0	0.0	0.0	0.0

Frequenza delle parole

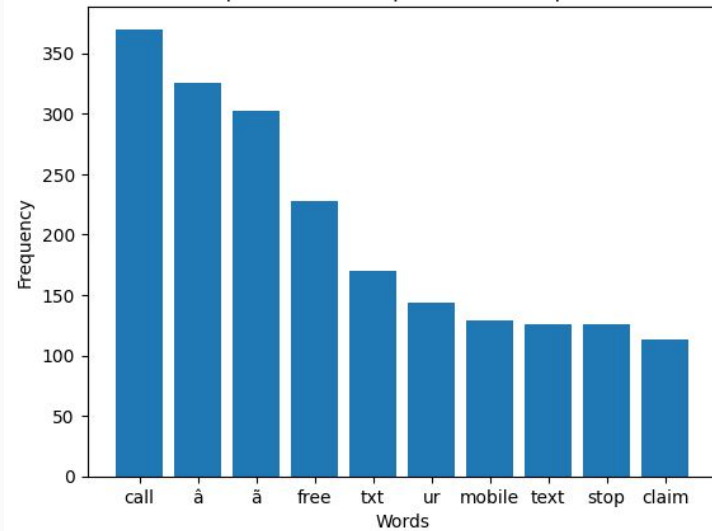
- Qui vengono riportate le parole più frequenti trovate nei vari sms.
- Facciamo vedere le differenze tra: le frequenze del dataset prima e dopo il sottocampionamento.



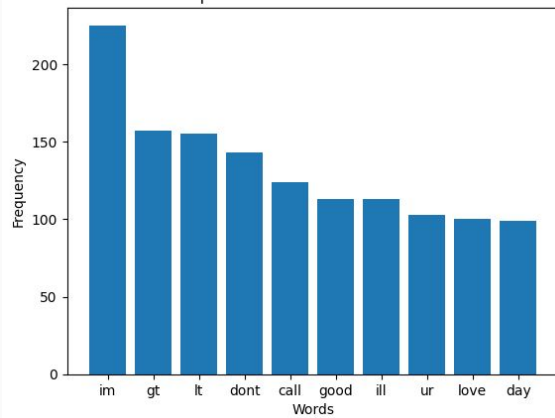
Most frequent words in spam in entire dataset



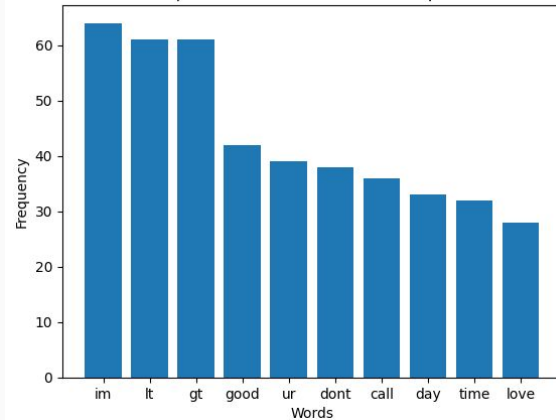
Most frequent words in spam in subsample dataset



Most frequent words in ham in entire dataset



Most frequent words in ham in subsample dataset



J48

- Abbiamo utilizzato J48 per costruire un albero di decisione per predire in base al contenuto se il messaggio è spam o meno.

weka.gui.GenericObjectEditor

weka.classifiers.trees.J48

Class for generating a pruned or unpruned C4.

More

Capabilities

batchSize 100

binarySplits False

collapseTree True

confidenceFactor 0.25

debug False

doNotCheckCapabilities False

doNotMakeSplitPointActualValue False

minNumObj 2

numDecimalPlaces 2

numFolds 3

reducedErrorPruning False

saveInstanceData False

seed 1

subtreeRaising True

unpruned False

useLaplace False

useMDLcorrection True

Test options

☐ Use training set

☐ Supplied test set Set...

☒ Cross-validation Folds 5

☐ Percentage split % 66

More options...

Risultati J48

```
Correctly Classified Instances      1352           90.4953 %
Incorrectly Classified Instances    142           9.5047 %
Kappa statistic                    0.8099
Mean absolute error                 0.1497
Root mean squared error             0.2873
Relative absolute error             29.9392 %
Root relative squared error         57.4612 %
Total Number of Instances          1494
```

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,926	0,116	0,888	0,926	0,907	0,811	0,924	0,875	ham
	0,884	0,074	0,923	0,884	0,903	0,811	0,924	0,937	spam
Weighted Avg.	0,905	0,095	0,906	0,905	0,905	0,811	0,924	0,906	

=== Confusion Matrix ===

```
  a  b  <-- classified as
692 55 |  a = ham
 87 660 | b = spam
```

con pruning

```
Correctly Classified Instances      1368           91.5663 %
Incorrectly Classified Instances    126           8.4337 %
Kappa statistic                    0.8313
Mean absolute error                 0.1354
Root mean squared error             0.2707
Relative absolute error             27.0755 %
Root relative squared error         54.1451 %
Total Number of Instances          1494
```

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,963	0,131	0,880	0,963	0,919	0,835	0,936	0,895	ham
	0,869	0,037	0,959	0,869	0,912	0,835	0,936	0,950	spam
Weighted Avg.	0,916	0,084	0,919	0,916	0,915	0,835	0,936	0,923	

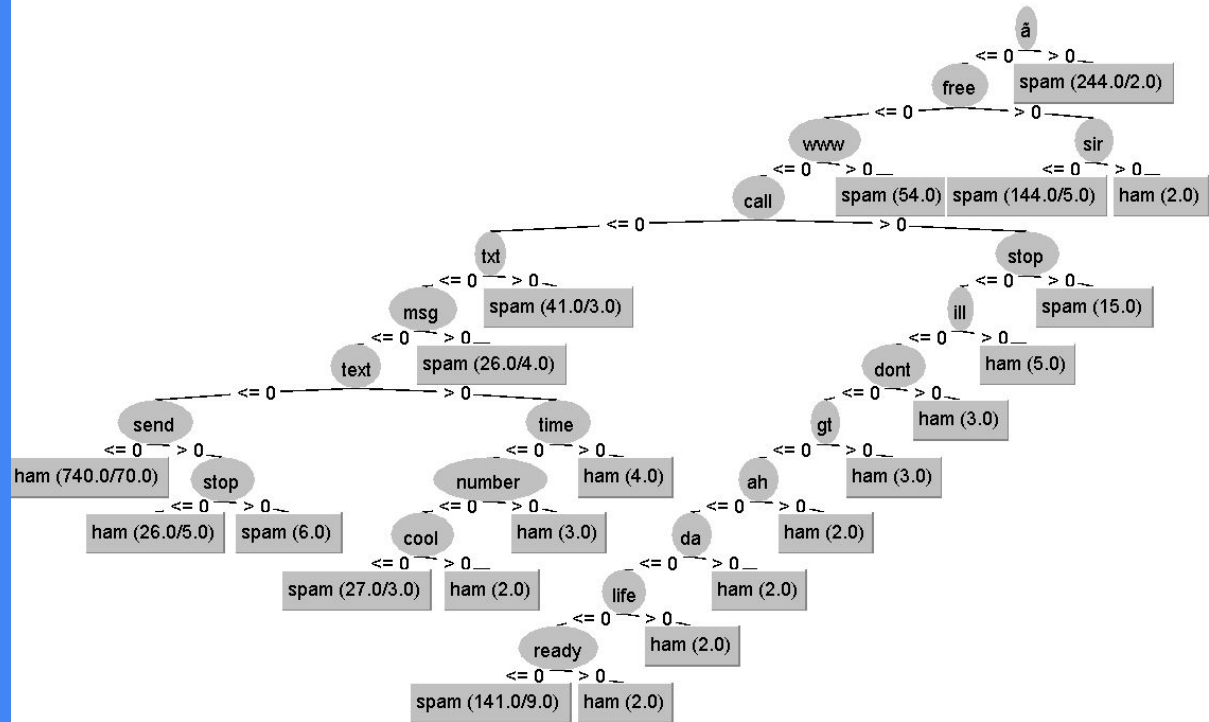
=== Confusion Matrix ===

```
  a  b  <-- classified as
719 28 |  a = ham
 98 649 | b = spam
```

senza pruning

Albero di decisione

- Riportiamo qua a destra l'albero di decisione generato (è stato effettuato il pruning)



Rules Classifier JRIP

- Proviamo adesso ad utilizzare l'approccio con JRIP

The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.rules.JRIP' classifier. The window has a title bar with a close button. Below the title bar, the class name 'weka.classifiers.rules.JRIP' is displayed. The main area is divided into an 'About' section and a configuration section. The 'About' section contains a text description of the classifier and two buttons: 'More' and 'Capabilities'. The configuration section contains several parameters, each with a label and a value field. The parameters are: 'batchSize' (100), 'checkErrorRate' (True), 'debug' (False), 'doNotCheckCapabilities' (False), 'folds' (3), 'minNo' (2.0), 'numDecimalPlaces' (2), 'optimizations' (2), 'seed' (10), and 'usePruning' (True). At the bottom of the window, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

weka.gui.GenericObjectEditor

weka.classifiers.rules.JRIP

About

This class implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed by William W.

More

Capabilities

batchSize 100

checkErrorRate True

debug False

doNotCheckCapabilities False

folds 3

minNo 2.0

numDecimalPlaces 2

optimizations 2

seed 10

usePruning True

Open... Save... OK Cancel

Regole JRIP

- A fianco le regole generate dall'algoritmo
- Così come nel modello di classificazione JRIP, anche qui notiamo come alcune parole vengono considerate fonte di probabile messaggio di spam...

(call <= 0) and (txt <= 0) and (â <= 0) and (free <= 0) and (www <= 0) and (text <= 0) and (send <= 0) and (reply <= 0) and (service <= 0) and (http <= 0) and (ringtone <= 0) and (chat <= 0) and (im >= 1)
=> Class=ham (53.0/0.0)

(call <= 0) and (txt <= 0) and (text <= 0) and (send <= 0) and (reply <= 0) and (uk <= 0) and (â <= 0) and (http <= 0) and (service <= 0) and (sms <= 0) and (chat <= 0) and (ringtone <= 0) and (free <= 0)
=> Class=ham (646.0/38.0)

(â <= 0) and (send >= 1) and (stop <= 0) and (row <= 0) and (pounds <= 0) and (mobile <= 0) and (eve <= 0) and (auction <= 0)
=> Class=ham (28.0/4.0)

(ã <= 0) and (ill >= 1) => Class=ham (9.0/1.0)

(ã <= 0) and (gt >= 1) => Class=ham (4.0/0.0)
=> Class=spam (754.0/50.0)

Risultati JRIP

```
Correctly Classified Instances    1358          90.8969 %
Incorrectly Classified Instances   136          9.1031 %
Kappa statistic                   0.8179
Mean absolute error               0.1385
Root mean squared error           0.2873
Relative absolute error           27.7026 %
Root relative squared error       57.45 %
Total Number of Instances        1494
```

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,917	0,099	0,903	0,917	0,910	0,818	0,919	0,892	ham
	0,901	0,083	0,916	0,901	0,908	0,818	0,919	0,909	spam
Weighted Avg.	0,909	0,091	0,909	0,909	0,909	0,818	0,919	0,900	

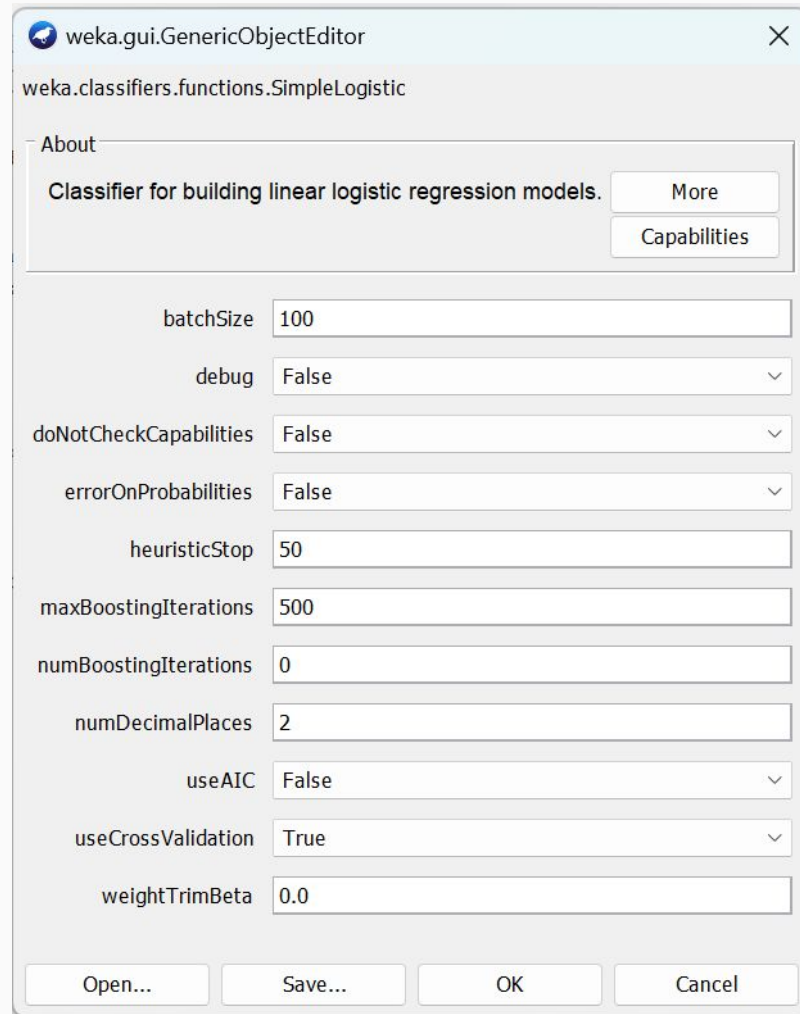
=== Confusion Matrix ===

```
  a   b  <-- classified as
685 62 |  a = ham
 74 673 |  b = spam
```

- Raggiungiamo una accuracy del 90.89%

LogitBoost

- Proviamo adesso ad utilizzare l'approccio usando SimpleLogistic, che sfrutta il boosting di LogitBoost, il quale utilizza come "base learners" delle regressioni.



The screenshot shows the 'weka.gui.GenericObjectEditor' window with the title bar. The main content area displays the class 'weka.classifiers.functions.SimpleLogistic'. Below this, there is an 'About' section with the text 'Classifier for building linear logistic regression models.' and two buttons: 'More' and 'Capabilities'. The main settings area contains several parameters, each with a label and a value field:

Parameter	Value
batchSize	100
debug	False
doNotCheckCapabilities	False
errorOnProbabilities	False
heuristicStop	50
maxBoostingIterations	500
numBoostingIterations	0
numDecimalPlaces	2
useAIC	False
useCrossValidation	True
weightTrimBeta	0.0

At the bottom of the window, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

Logistic regression con LogitBoost (Weka SimpleLogistic)

- L'algoritmo utilizza dei pesi.
- Il peso di grandezza massima si ha quando $p(x) = p(1-x)$

LogitBoost (J classes)

1. Start with weights $w_{ij} = 1/n$, $i = 1, \dots, n$, $j = 1, \dots, J$, $F_j(x) = 0$
and $p_j(x) = 1/J \quad \forall j$

2. Repeat for $m = 1, \dots, M$:

(a) Repeat for $j = 1, \dots, J$:

i. Compute working responses and weights in the j th class

$$z_{ij} = \frac{y_{ij}^* - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))}$$

$$w_{ij} = p_j(x_i)(1 - p_j(x_i))$$

ii. Fit the function $f_{mj}(x)$ by a weighted least-squares regression
of z_{ij} to x_i with weights w_{ij}

(b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J} \sum_{k=1}^J f_{mk}(x))$, $F_j(x) \leftarrow F_j(x) + f_{mj}(x)$

(c) Update $p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^J e^{F_k(x)}}$

3. Output the classifier $\operatorname{argmax}_j F_j(x)$

Risultati LogitBoost

Time taken to build model: 19.89 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	1405	94.0428 %
Incorrectly Classified Instances	89	5.9572 %
Kappa statistic	0.8809	
Mean absolute error	0.0978	
Root mean squared error	0.2185	
Relative absolute error	19.5617 %	
Root relative squared error	43.701 %	
Total Number of Instances	1494	

=== Detailed Accuracy By Class ===

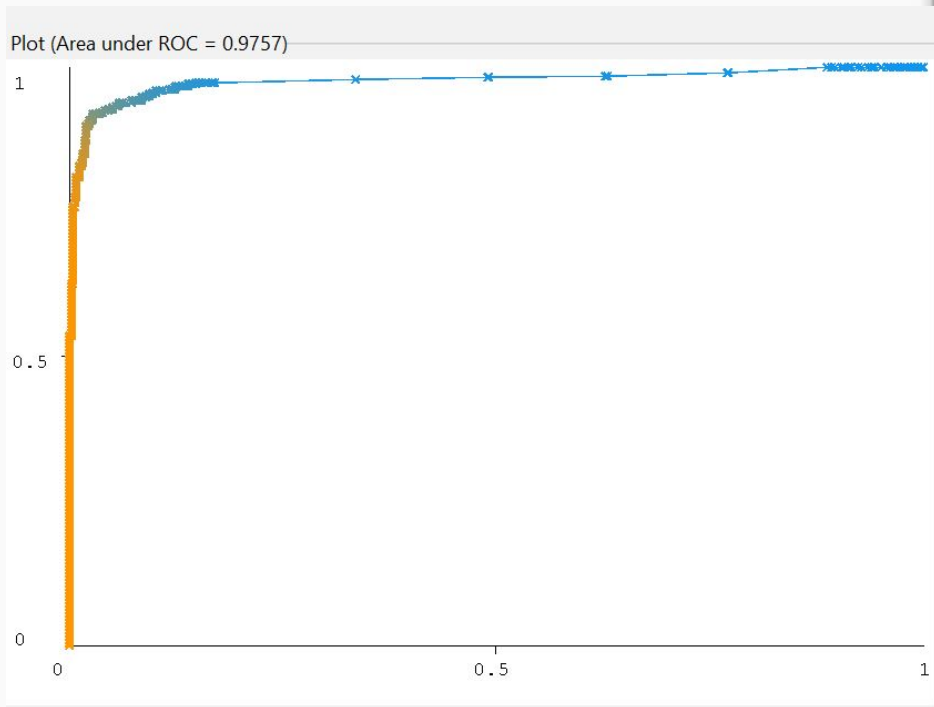
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,975	0,094	0,912	0,975	0,942	0,883	0,976	0,962	ham
	0,906	0,025	0,973	0,906	0,938	0,883	0,976	0,981	spam
Weighted Avg.	0,940	0,060	0,942	0,940	0,940	0,883	0,976	0,972	

=== Confusion Matrix ===

```
a  b  <-- classified as
728 19 | a = ham
 70 677 | b = spam
```

- Raggiungiamo una accuracy ancora più elevata, del 94%!

ROC LogitBoost



- L'area sotto la curva ROC è vicina ad 1!

AdaboostM1

- Usiamo l'algoritmo AdaboostM1.
- Come classificatore utilizziamo il J48

The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.meta.AdaBoostM1' class. The 'About' tab is active, displaying the description: 'Class for boosting a nominal class classifier using the Adaboost M1 method.' There are 'More' and 'Capabilities' buttons. Below this, various parameters are configured in a list:

- batchSize: 100
- classifier: Choose **J48 -C 0.25 -M 2** (The 'Choose' button is highlighted with a red dashed box)
- debug: False
- doNotCheckCapabilities: False
- numDecimalPlaces: 2
- numIterations: 10
- resume: False
- seed: 1
- useResampling: False
- weightThreshold: 100

At the bottom, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

Risultati Adaboost

```
Correctly Classified Instances      1399           93.6412 %
Incorrectly Classified Instances     95           6.3588 %
Kappa statistic                     0.8728
Mean absolute error                  0.062
Root mean squared error              0.2413
Relative absolute error              12.4005 %
Root relative squared error          48.2558 %
Total Number of Instances          1494
```

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,949	0,076	0,926	0,949	0,937	0,873	0,972	0,954	ham
	0,924	0,051	0,948	0,924	0,936	0,873	0,972	0,979	spam
Weighted Avg.	0,936	0,064	0,937	0,936	0,936	0,873	0,972	0,966	

=== Confusion Matrix ===

```
 a   b  <-- classified as
709 38 |  a = ham
 57 690 | b = spam
```

- Raggiungiamo una accuracy del 93.64%

Random Forest

- Infine proviamo ad utilizzare le random forest.
- Per valutare l'errore sfruttiamo le previsioni out of bag.

The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.trees.RandomForest' classifier. The 'About' section describes it as a 'Class for constructing a forest of random trees.' and includes 'More' and 'Capabilities' buttons. Below this, various parameters are configured in a list:

- bagSizePercent: 100
- batchSize: 50
- breakTiesRandomly: True
- calcOutOfBag: True
- computeAttributeImportance: False
- debug: False
- doNotCheckCapabilities: False
- maxDepth: 0
- numDecimalPlaces: 2
- numExecutionSlots: 1
- numFeatures: 0
- numIterations: 100
- outputOutOfBagComplexityStatistics: True
- printClassifiers: True
- seed: 1
- storeOutOfBagPredictions: True

At the bottom, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

Risultati Random Forest

*** Out-of-bag estimates ***

Correctly Classified Instances	1443	96.5863 %
Incorrectly Classified Instances	51	3.4137 %
Kappa statistic	0.9317	
K&B Relative Info Score	82.4146 %	
K&B Information Score	1231.2736 bits	0.8241 bits/instance
Class complexity order 0	1494 bits	1 bits/instance
Class complexity scheme	3502.7266 bits	2.3445 bits/instance
Complexity improvement (Sf)	-2008.7266 bits	-1.3445 bits/instance
Mean absolute error	0.1002	
Root mean squared error	0.1878	
Relative absolute error	20.0387 %	
Root relative squared error	37.5629 %	
Total Number of Instances	1494	

=== Summary ===

Correctly Classified Instances	1494	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0.0371		
Root mean squared error	0.0709		
Relative absolute error	7.4257 %		
Root relative squared error	14.1702 %		
Total Number of Instances	1494		

- Raggiungiamo una accuracy del 96.58%
- In fase di training otteniamo il 100% di accuracy

Considerazioni Finali

- Nel nostro caso ci interessa una accuratezza più elevata e allo stesso tempo un numero di falsi positivi basso (bisognerebbe evitare che un sms importante venga classificato erroneamente come spam...)
- Per le configurazioni provate, l'algoritmo RandomForest di Weka performa meglio.

	Accuracy	FP
J48 (unpruned)	91.56%	28
LogitBoost	94.04%	19
JRip	90.89%	62
Adaboost M1	93.64%	38
Random Forest	96.58%	

Appendice:

MultinomialNB

- Per curiosità abbiamo usato una MultinomialNB di scikit-learn.
- Tale funzione utilizza un Naive Bayes con la correzione di Laplace (se $\alpha = 1$)
- Riusciamo con k-fold = 5 ad avere un accuracy del 95%! Inoltre Naive Bayes come algoritmo è particolarmente veloce e molto utilizzato anche per questo nella spam detection.

```
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.naive_bayes import MultinomialNB

X = dataframe_sub.drop("Class",axis="columns")
y = dataframe_sub["Class"].apply(lambda x: int(x == "spam"))

#alpha = 1 -> Laplace
mnb = MultinomialNB(alpha=1)
scores = cross_val_score(mnb, X, y, cv=5)
print("%.3f accuracy and standard error %.3f" % (scores.mean(), scores.std()))
```

0.952 accuracy and standard error 0.010

Bibliografia

- <https://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html>
- Machine Learning, 59, 161–205, 2005 2005
Springer Science + Business Media, Inc.
Manufactured in The Netherlands. Logistic
Model Trees* NIELS LANDWEHR
landwehr@informatik.uni-freiburg.de Institute
for Computer Science, University of Freiburg,
Freiburg, Germany MARK HALL
mhall@cs.waikato.ac.nz EIBE FRANK
eibe@cs.waikato.ac.nz Department of
Computer Science, University of Waikato,
Hamilton, New Zealand
- Experiments with a New Boosting Algorithm,
Yoav Freund Robert E. Schapire
- <https://oneapi-src.github.io/oneDAL/daal/algorithms/boosting/logitboost.html>