Progetto High Performance & Scalable Analytics

SMS Spam Collection Dataset

Modello di classificazione su database distribuito

Lorenzo Basile, Ada Maragno, Giangaetano Pachera, Gianluca Sperduti

Master in Big Data Analytics & Social Mining

A.A. 2020/202

Abstract

SMS Spam Collection è un dataset disponibile su Kaggle dal 2016, utile per addestrare modelli per la classificazione di spam sms.
Il dataset contiene 5574 SMS in lingua inglese, ognuno etichettato dalla sua classe Spam o Ham (legittimo).
Sono presenti due colonne: la colonna v1 indicante la classe, la colonna v2 il contenuto testuale del SMS.
Questo progetto vuol creare un modello di classificazione in grado di riconoscere se un SMS può esseere considerato Spam o Ham, limitandosi all'analisi testuale.
L'esplorazione, l'analisi e lo studio dei modelli di classificazione verranno svolti su un database distribuito utilizzando la libreria PySpark di Python.
Il progetto è suddiviso in tre fasi:

1. Data exploration, cleaning, understanding, engineering
2. Pre-processing
3. Modeling

1.1 Data exploration and Data cleaning

Il dataset, fornito in formato csv, ha richiesto una ri-codifica nella standratd UTF-8 per via di UnicodeDecodeError.

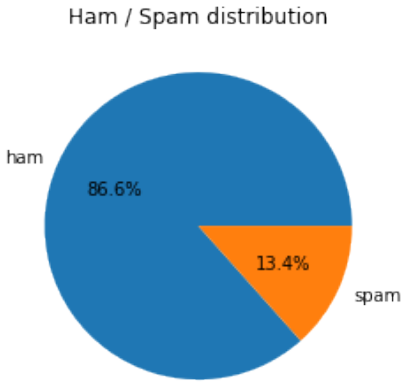
Il contenuto del dataset, proveniente da diverse fonti, presenta una formattazione del testo non omogenea, e spesso i contenuti presentano caratteri speciali di chiusura del testo.
A causa della presenza di segni di punteggiatura nel contenuto testuale, è stato necessario modificare il csv delimter "," in "<", poiché il primo portava un conflitto nel mapping con la funzione .map(lambda x: x.split('","')).
Il risultato del grouping per classe si presentava inizialmente così:

```
[('ham"""', 2), ('spam', 747), ('ham', 4825)]
```

Alcuni record hanno richiesto quindi un intervento di replacement del valore della classe.

1.2 Data understanding and Data Engineering

Nel seguente pie chart viene riportata la distribuzione degli sms per classe. È evidente uno sbilanciamento del dataset, con 747 record Spam e 4827 record Ham.

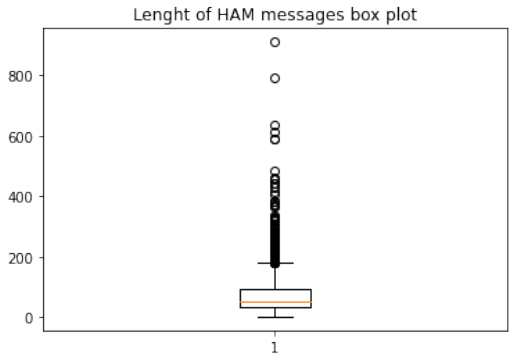


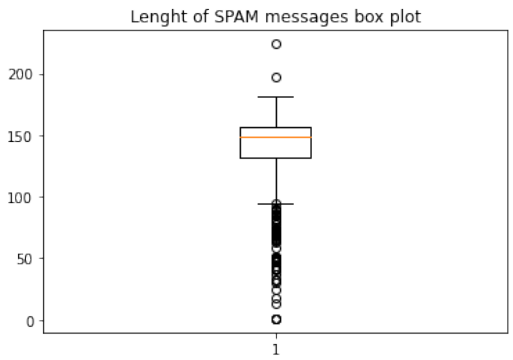
L'analisi si è concentrata sulla sintassi e contenuto del testo, creando 9 feature utilizzate successivamente nella modellazione. Queste sono: lunghezza del testo, conteggio della punteggiatura, conteggio dei punti interrogativi, conteggio

dei punti esclamativi, conteggio dei caratteri maiuscoli, conteggio delle parole maiuscole, presenza di link, conteggio di numeri, tokenizzazione con rimozione di stopwords, most frequent word per classe.

Feature 1: lunghezza del testo

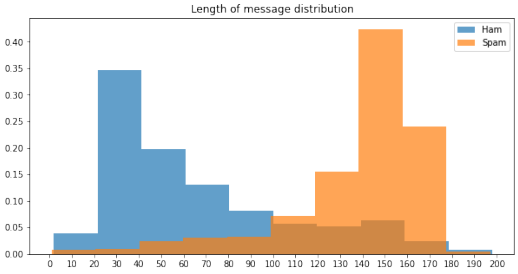
La prima analisi si è concentrata sulla distribuzione della lunghezza degli SMS per classe. Nei seguenti boxplot vengono descritte le caratteristiche salienti della distribuzione della lunghezza.





Nel seguente istogramma vengono confrontate quindi le due classi per lunghezza di SMS, senza considerare gli SMS con lunghezza superiore a 200 caratteri, definiti questi come outliers.

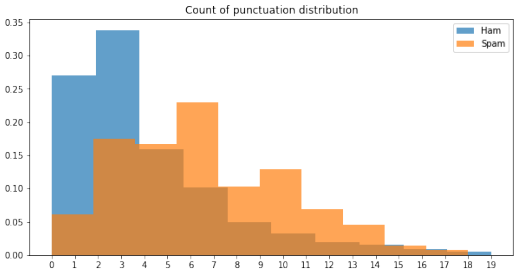
I due set vengono confrontati per lunghezza di SMS nel seguente istogramma:



È evidente una grande concentrazione di Spam SMS a 160 caratteri, lunghezza massima consentita per SMS dovuta alla codifica a 7 bit utilizzata solo con il set di caratteri di base GSM.[1]

Feature 2: caratteri di punteggiatura e caratteri speciali

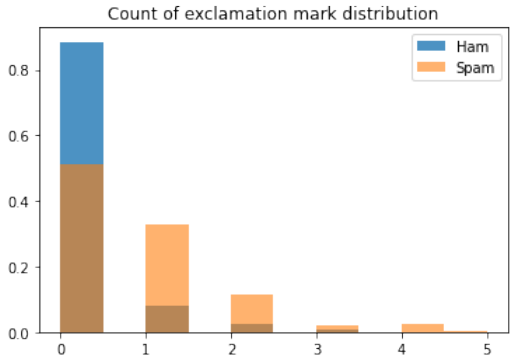
La successiva analisi si è concentrata sulla presenza di segni di punteggiatura, nonché tutti quei caratteri presenti nella tabella ASCII diversi da lettere, numeri e carattere spazio.

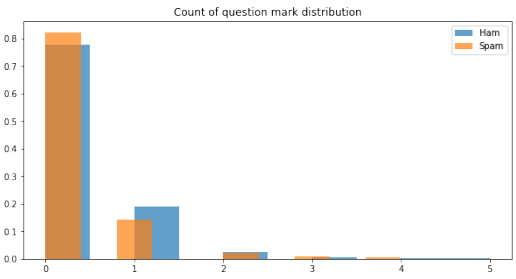


Feature 3-4: punti interrogativi e punti esclamativi

Le feature vogliono evidenziare la presenza di punti interrogativi e punti esclamativi contandone il numero di caratteri per ogni SMS. L'istogramma mostra in misura percentuale, per ogni set, la distribuzione di punti esclamativi (Figura 1) e punti interrogativi (Figura 2).

[1] https://websms.it/blog/news-trends/codifica-sms

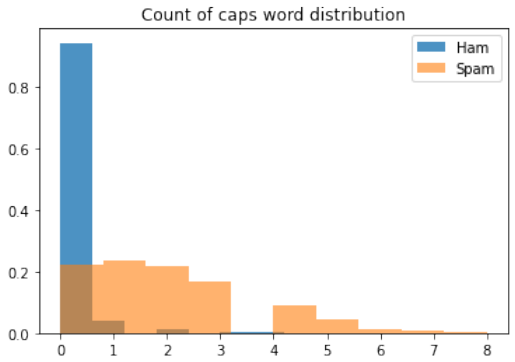


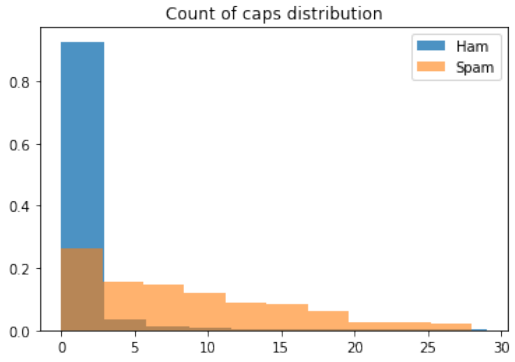


È evidente una significativa presenza di punti esclamativi nel set di Spam SMS, mentre risulta equa la presenza di punti interrogativi in entrambi i set.

Feature 5-6: caratteri e parole in maiuscolo

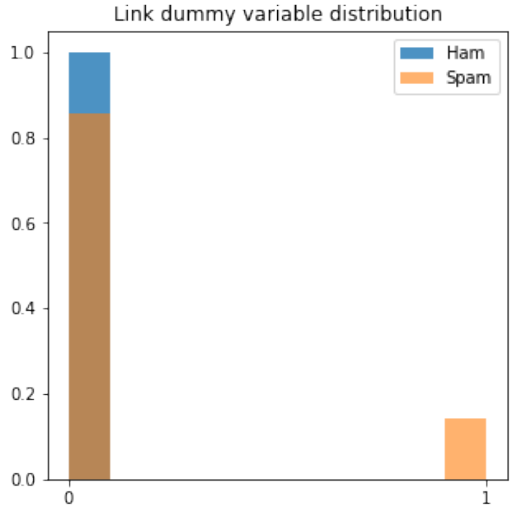
Le feature rappresentano rispettivamente il conteggio di lettere e di parole in maiuscolo. È evidente una forte presenza di caratteri e parole in maiuscolo nel set Spam.





Feature 7: presenza di link

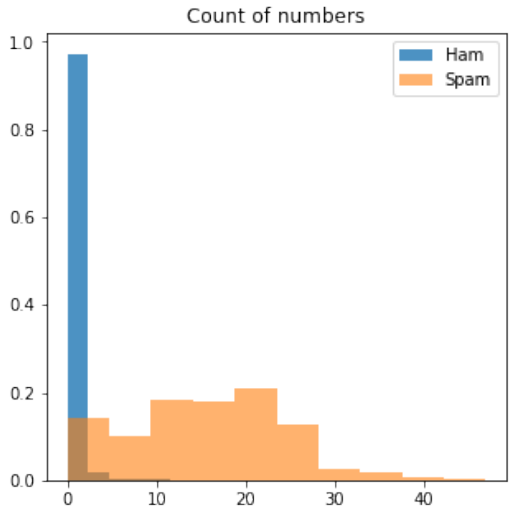
La presenza di link viene rappresentata da una variabile dicotomica 0,1 la quale evidenza la presenza di caratteri nella forma “http” o “www”.



Come evidenzia l’istogramma, nessun record Ham presenta link, mentre il 17% dei record Spam presentano almeno un link.

Feature 8: presenza di numeri

La presenza di numeri viene rappresentata con il conteggio di caratteri numerici all’interno del testo. Il seguente grafico mostra in misura percentuale la distribuzione del conteggio dei numeri nel dataset.



È evidente una significativa presenza di caratteri numerici nel set Spam.

Feature 9: bag of words

È stata definita la funzione BagOfWords, la quale restituisce token di parole escludendo le Stopwords dal set di default in lingua inglese della libreria NLTK e da un set di parole utilizzate come abbreviazioni di stopwords {'ur','n','u','r'}, tipiche degli SMS. Questa variabile è stata creata avendo in mente l'utilizzo dell'Hashing TF in fase di classificazione, strategia successivamente abbandonata a favore dell'utilizzo delle due variabili illustrate di seguito, per i motivi riportati nel successivo capitolo.

Feature 10-11: most frequent word per classe

Definiti due set di parole Ham e Spam, contenenti le 25 parole più frequenti per ogni classe, le feature 10 e 11 contano la presenza di parole presenti nel Training set, creato in fase di pre-processing.

Ham Set:

[('gt', 274), ('lt', 272), ('get', 251), ('ok', 231), ('go', 205), ('call', 193), ('good', 193), ('know', 192), ('got', 185), ('like', 182), ('come', 179), ('day', 177), ('time', 160), ('love', 157), ('want', 139), ('one', 133), ('da', 131), ('lor', 130), ('home', 129), ('going', 127), ('need', 126), ('still', 124), ('sorry', 118), ('today', 115), ('k', 114)]

Spam set:

[('call', 290), ('free', 192), ('txt', 126), ('mobile', 108), ('stop', 95), ('claim', 93), ('text', 86), ('reply', 83), ('www', 81), ('prize', 74), ('get', 73), ('uk', 62), ('new', 61), ('send', 57), ('nokia', 56), ('cash', 55), ('week', 54), ('tone', 53), ('urgent', 52), ('150p', 49), ('co', 48), ('win', 46), ('c', 46), ('please', 45), ('contact', 45)]

1.3 Classification pre-processing

Nella fase di pre-processing si è voluto creare due vettori LabeledPoint.

Una funzione creata appositamente per convertire il contenuto in uno Sparse vector ci ha permesso di rappresentare in un unico vettore sparse i Labeled Point delle prime nove feature, oltre a quelle relative all'HashingTF.

Un secondo vettore Labeled Point ci ha permesso di riassumere invece le dieci feature precedentemente create, senza tener conto della feature HashingTF.

Un primo tentativo di modellazione ha cercato di tenere conto di tutte le features generate dal processo di Hashing TF settato con un valore di 32768. Dato l'eccessivo costo di calcolo e gli scarsi risultati ottenuti, si è deciso di escludere la funzione HashingTF dai successivi modelli, tenendo quindi in considerazione il solo Labeled Point rappresentante le dieci feature descritte poc'anzi.

È stato creato quindi un Training Set e un Test Set, nel rapporto 80:20.

1.4 Modeling

In prima fase è stato eseguito un primo giro di modelli al fine di valutarne la performance generale. Per ogni modello sono stati valutati gli indicatori di precision, accuracy e recall, nonché l'errore di previsione sul training test e sul test set. Sebbene l'obiettivo principale sia quello di ottenere la migliore performance possibile, altro criterio di scelta è stato quello di preferire modelli con una alta precisione sulla classe 1 (spam) dato il contesto del problema, ovvero che un messaggio non automatizzato possa essere bloccato dall'algoritmo.

Sono stati applicati i modelli di

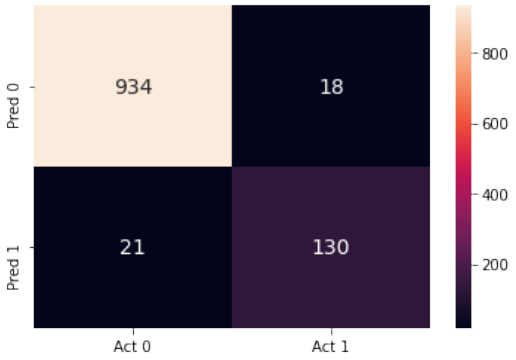
- Logistic Regression
- Decision Tree
- Random forest
- Gradient boost classifier

Di seguito viene riportato l'esito di ogni singolo modello.

Logistic regression

Il modello di regressione logistica si è mostrato meno affidabile rispetto agli altri, sebbene mostri comunque delle buone performance. Questo può essere dovuto a diverse cause, tra le quali la potenziale correlazione tra feature e la presenza di molti outliers nel dataset.

```
Train Error = 0.0308
Test Error = 0.0380

Train set:
Accuracy: 0.9691
Precision: 0.9113
Recall: 0.8538

Test set:
Accuracy: 0.9619
Precision: 0.8786
Recall: 0.8311
```



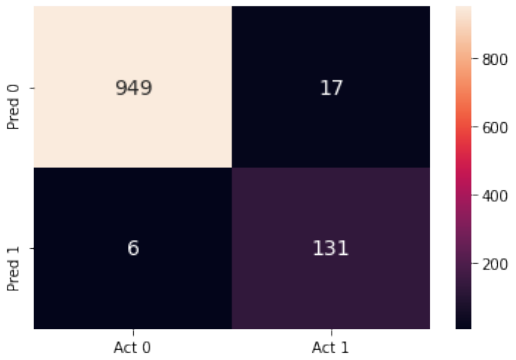
Decision tree

Il modello decision tree è stato generato con i seguenti iperparametri:

```
impurity='gini', maxDepth=5, maxBins=32
```

Il decision tree si mostra più performante rispetto al modello logistico e più in generale i risultati ottenuti sono soddisfacenti.

```
Train Error = 0.0125
Test Error = 0.0208

Train set:
Accuracy: 0.9875
Precision: 0.9773
Recall: 0.9286

Test set:
Accuracy: 0.9791
Precision: 0.9562
Recall: 0.8851
```



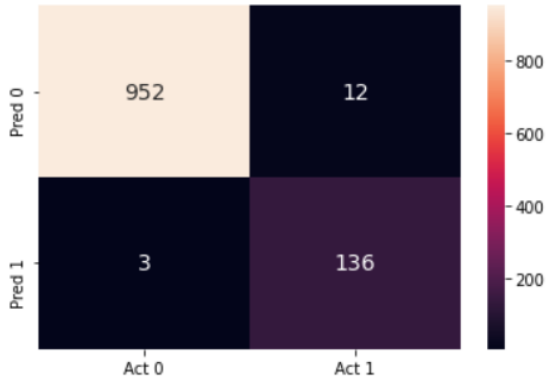
Random Forest

Il modello Random Forest è stato generato con i seguenti iperparametri:

impurity='gini', numTrees=54,
maxDepth=6, maxBins=100

Seppur di poco, il modello Random Forest si mostra migliore del precedente in termini di performance per tutti gli indicatori.

```
Train Error = 0.0091
Test Error = 0.0135

Train set:
Accuracy: 0.9908
Precision: 1.0
Recall: 0.9318

Test set:
Accuracy: 0.9864
Precision: 0.9784
Recall: 0.9189
```



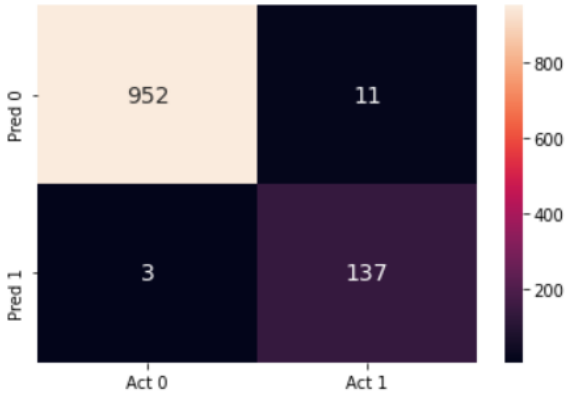
Gradient Boost Classifier

Il modello Gradient Boost è stato generato con i seguenti iperparametri:

numIterations=54

Anche questo modello si dimostra superiore rispetto ai primi due, dati i livelli di performance di seguito riportati.

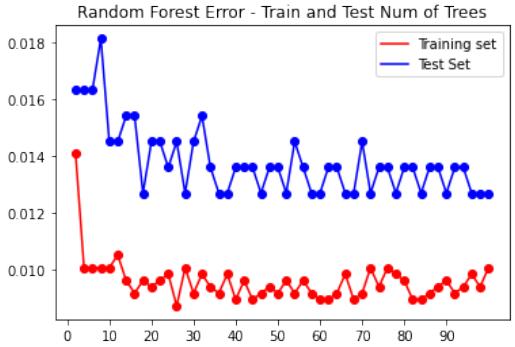
```
Train Error = 0.0105
Test Error = .01269

Train set:
Accuracy: 0.995
Precision: 0.9894
Recall: 0.9318

Test set:
Accuracy: 0.9873
Precision: 0.9786
Recall: 0.9257
```



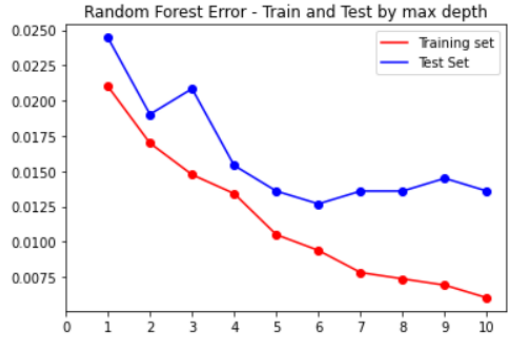
Osservati i primi risultati, abbiamo deciso di scegliere i modelli con le migliori performance al fine di trovare il migliore settaggio degli iperparametri. Segue, quindi, una ultima fase di ottimizzazione per i modelli prescelti, ovvero Random forest e Gradient Boost Classifier.

Random Forest - ottimizzazione

Per il modello random forest sono stati generati 50 modelli con numTrees da 2 a 102, a step di 2. Per ogni foresta generata è stato riportato nel grafico seguente l'errore sul training set e sul test set, al fine di individuare il miglior numero di numTrees. Si può notare che superati i 90 alberi l'errore sul test set converge, per cui il numero di alberi scelto è ricaduto sul valore 92.



Dato il numero di alberi si è cercato il valore ottimale per il parametro di profondità degli alberi.

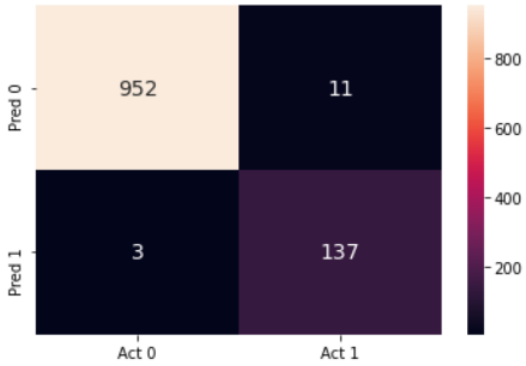


Dal precedente grafico si può osservare che il miglior valore per l'iperparametro in oggetto sia 6.

Il modello è stato quindi settato con i seguenti iperparametri:

impurity='gini', numTrees=92, maxDepth=6, maxBins=100

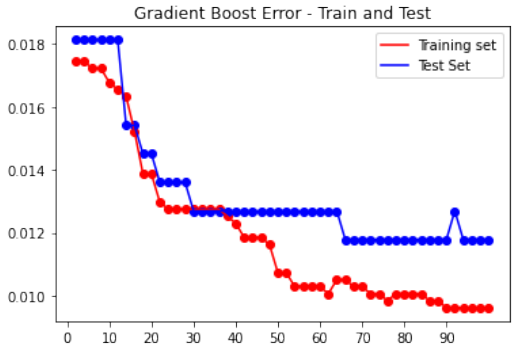
```
Train Error = 0.0093
Test Error = 0.0126

Train set:
Accuracy: 0.9906
Precision: 0.9982
Recall: 0.9318

Test set:
Accuracy: 0.9873
Precision: 0.9786
Recall: 0.9257
```



Il modello mostra un leggero miglioramento della recall rispetto al precedente.

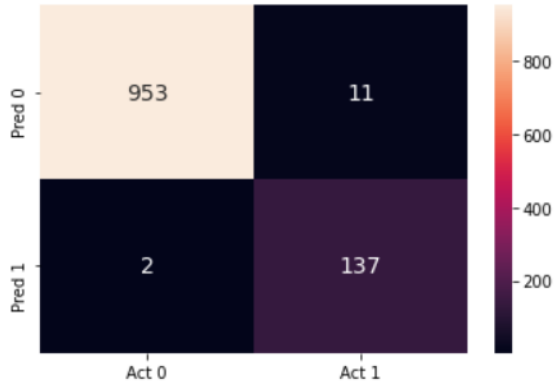
GB Classifier - ottimizzazione

Per il modello grandient boost classifier sono stati effettuate 50 iterazioni con variazioni dell'iperparametro numIterations da 2 a 102, a step di 2. Per ogni albero generato è stato riportato nel grafico seguente l'errore sul training set e sul test set, al fine di individuare il miglior numero di numIterations. Il valore migliore per numIterations è 64.



L’esito del gbc con numIterations = 64 è il seguente.

```
Train Error = 0.0100
Test Error = 0.0117

Train set:
Accuracy: 0.9899
Precision: 0.9947
Recall: 0.9301

Test set:
Accuracy: 0.9882
Precision: 0.9856
Recall: 0.9257
```



Seppure di poco, quest’ultimo modello si presenta come quello con la performance migliore ed è quindi quello di nostra scelta.