Progetto di Big Data e Business Intelligence

Classificazione mail di spam

1) Look at the big picture

Il task da svolgere è un task di classificazione supervisionato, l'obiettivo di questo progetto è quello di riuscire a trovare un modello di classificazione che riesca a predire se una mail sia una mail di spam oppure no.

Nella feature target le mail di spam sono indicate con il numero 1, mentre le altre con 0.

Andremo a misurare le performance con Recall, Precision ed F-1, accantoniamo subito l'idea di utilizzare l'Accuracy perché il dataset è sbilanciato.

L'idea generale è quella di tenere più in considerazione la Recall, perché come nell'esempio fatto a lezione, preferisco ricevere una mail di spam in più, piuttosto che non vedere una mail importante perché è finita nello spam.

2) Get the data

Le features presenti nel dataset sono complessivamente 3002, dove la prima identifica il numero di mail e l'ultima è la target.

Le altre features identificano ogni parola possibile presente all'interno della mail, ed ogni valore presente in ogni riga, identifica quante volte quella parola è comparsa nella mail.

Possiamo subito rimuovere la prima feature in quanto non ha alcun significato per la predizione, se non quello di identificare ogni mail.

Escludendo anche la feature target, restano 3000 feature sulle quali lavorare per trovare il modello.

A questo punto vediamo che le righe in totale sono 5172, rimuoviamo i duplicati e ci restano 4631 righe, notiamo di non avere valori negativi ne valori nulli, e notiamo che i dati presenti sono tutti di tipo INT.

Notiamo che solo il 31,55% delle mail nei dati è spam, mentre il restante 68,45% non è spam.

Splittiamo il dataset in training e test, 70% training e il restante 30% test.

```
Number of features:  3000
Number of rows:  4631
Number of negative values:  0
There are duplicated rows? ->  False
There are null values? ->  False
Label 0:  68.45 %
Label 1:  31.55 %
0    3170
1    1461
Name: Prediction, dtype: int64
```

3) Explore the data

La visualizzazione dei dati in questo progetto è un problema, perché avendo 3000 features sarebbe inadatto fare un istogramma per ogni feature, sarebbe quasi come un loop.

Quindi posticipiamo la data visualization dopo la riduzione della dimensionalità con PCA.

4) Prepare your data for ML algorithms

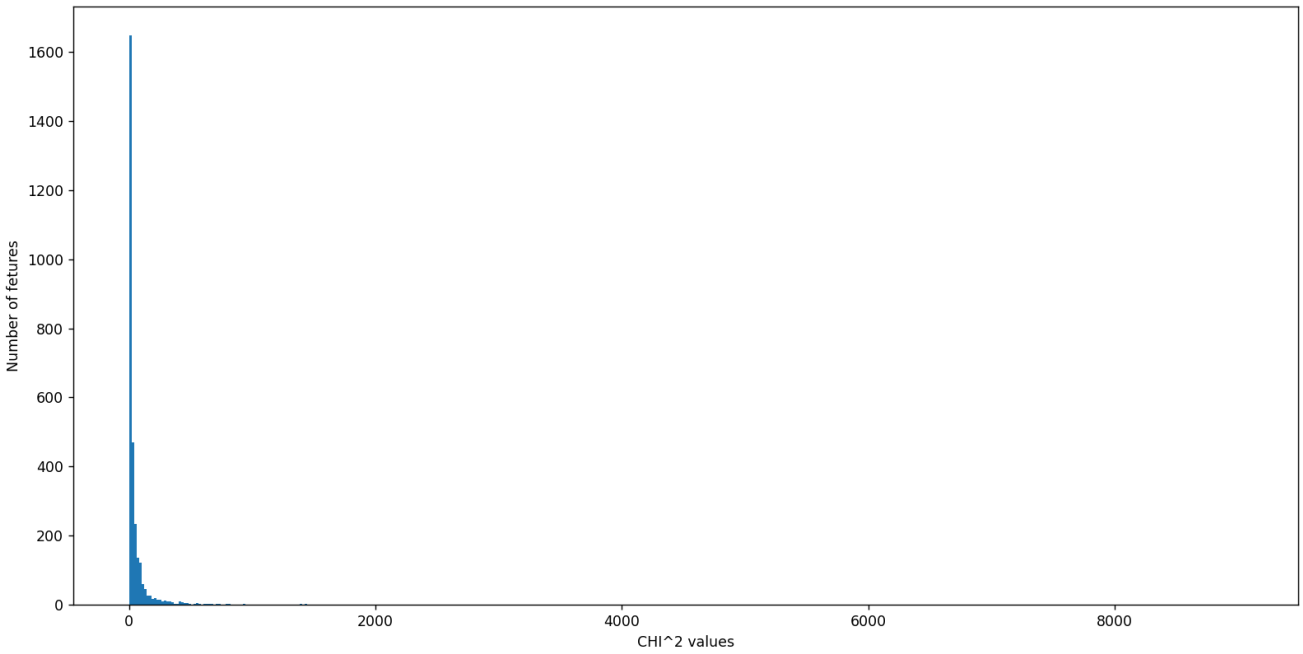
Come detto in precedenza, ho rimosso la feature "Email No." perché serve solo ed esclusivamente ad identificare la mail con un numero univoco.

Vediamo che i dati sono tutti di tipo INT, e che non ci sono valori mancanti o nulli.

Iniziamo la *Feature selection* con la misura di correlazione chi^2.

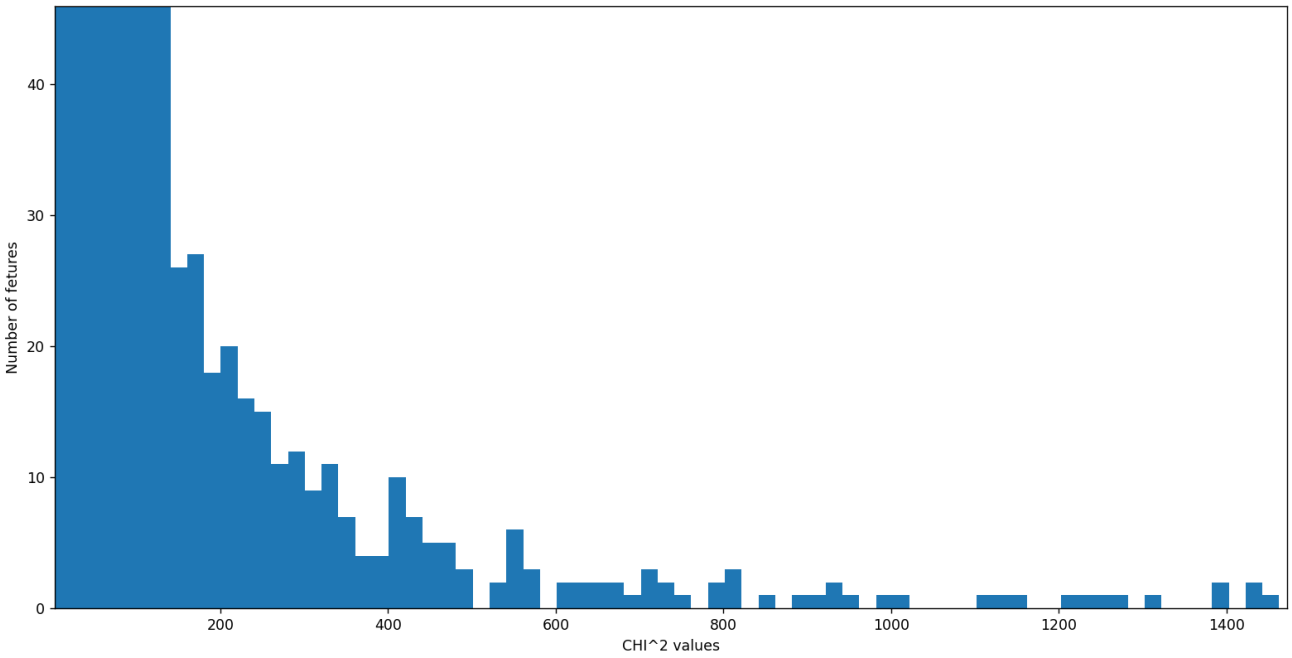
Utilizziamo la chi^2 perché abbiamo solo valori >=0 ed avendo molte feature, è molto più veloce rispetto alla Mutual Information.

Stampiamo e plottiamo i valori ottenuti dalle feature, e notiamo che moltissime features hanno una correlazione con la target molto bassa, pressoché 0.



Vengono visualizzate così distanti, prechè alcune features hanno valori di correlazione veramente alti.

Se eseguiamo uno zoom nel grafico, vedremo il seguente istogramma:



Stampiamo anche numericamente i risultati del chi^2:

```
CHI^2 RESULTS

THE MAXIMUM IS:
 Feature       zonedubai
Score      12745.507953
dtype: object

THE MINIMUM IS:
 Feature             a
Score       0.000006
dtype: object

THE AVG IS:  88.48779433476278
The features with chi2 <100 are:  2566
The features with chi2 between 101 e 500 are:  340
The features with chi2 between 501 e 1000 are:  53
The features with chi2 >1000 are:  39
```

A questo punto riduciamo drasticamente il numero di features utilizzate per addestrare il modello predittivo.

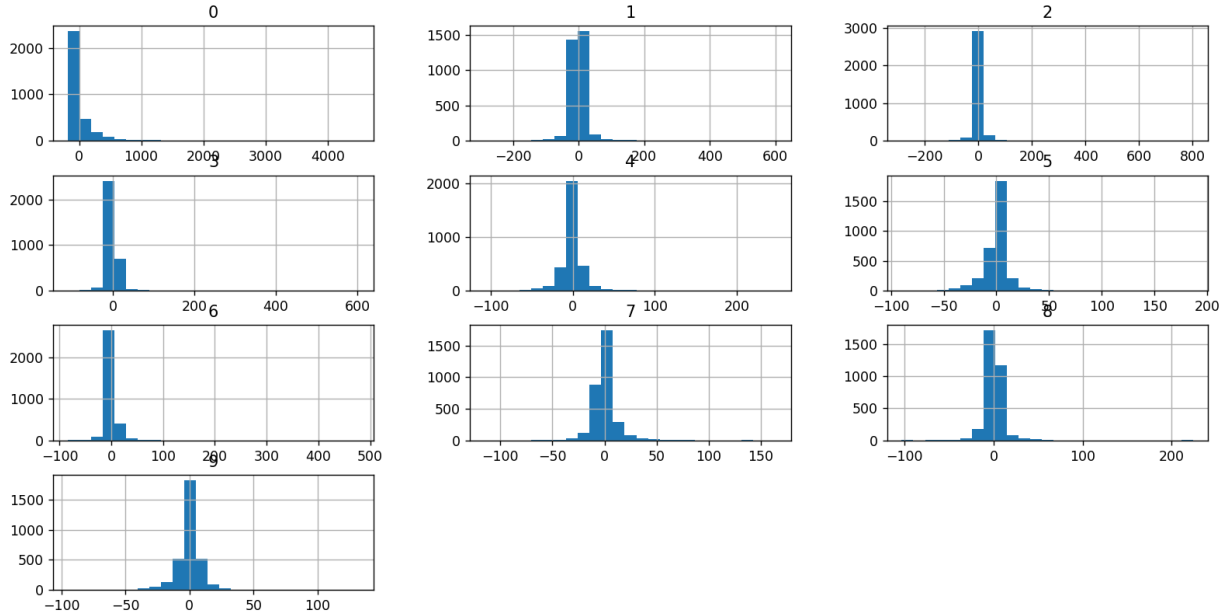
Ho pensato di poterlo fare perché le parole che hanno un valore di chi^2 molto basso, possono comparire sia in mail di spam che in mail normali.

Prendiamo ad esempio il caso della parola con chi^2 minore di tutte, cioè “a”, sappiamo che è una parola molto frequente nelle mail, quindi non può aiutarci a determinare se una mail è spam oppure no

Scegliamo le 50 features con i migliori risultati nel chi^2, subito dopo con queste 50 features faremo la riduzione della dimensionalità con PCA, mantenendo il 99% della varianza dei dati. (le nuove features saranno circa 10/12)

Se dovessimo scegliere più di 50 features, c'è un'alta possibilità di finire in overfitting.

A questo punto verrà eseguita una data visualization (Explore the data), un esempio di visualizzazione è questo:



La feature scaling non è stata fatta in quanto i valori sono tutti sulla stessa scala, partono tutti da 0.

5) Model selection

Essendo un task di classificazione, i 6 modelli che ho confrontato sono:

- Random forest classifier
- SVC
- Logistic regression
- Ada boost classifier
- Gradient boosting classifier
- XGB classifier

Tutti i modelli sono stati addestrati e comparati con una K-fold di 10 split, valutando ogni modello in base alla media di Recall, Precision ed F-1.

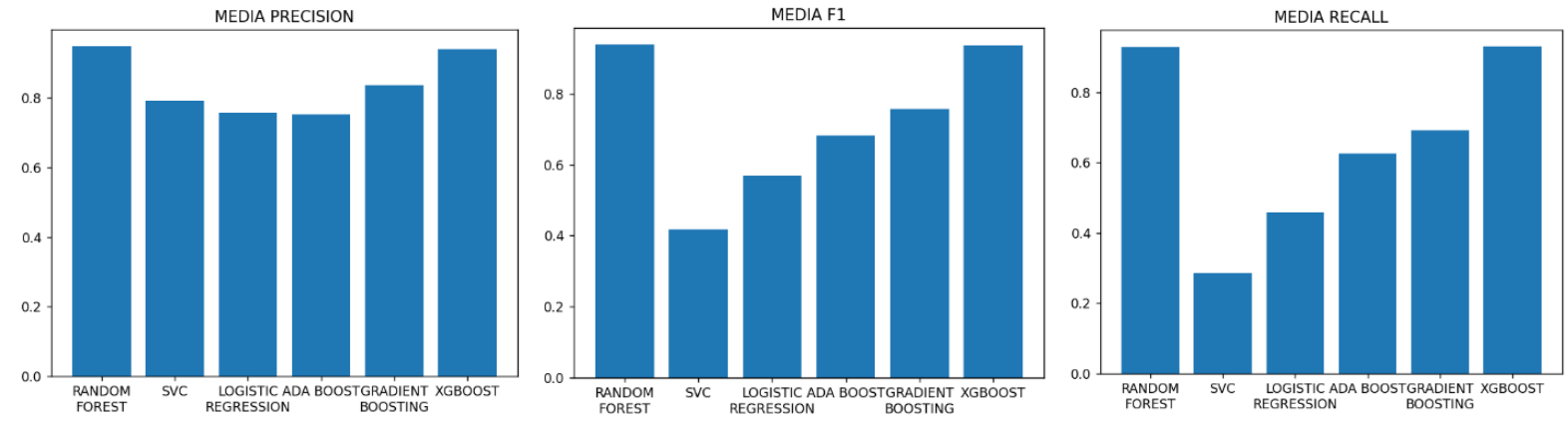
In ogni split della K-fold viene effettuata un'ulteriore suddivisione tra training e validation set.

Tutti i modelli sono stati utilizzati con i parametri di default.

Un esempio di risultati è il seguente:

```
RandomForestClassifier() :
Recall-> 0.9289795918367346
Precision-> 0.9490140114167099
F1-> 0.9382356225572306

SVC() :
Recall-> 0.28520489529815574
Precision-> 0.7915542950119707
F1-> 0.418725793272787

LogisticRegression() :
Recall-> 0.45890213519329964
Precision-> 0.7566000812453887
F1-> 0.570373937860397

AdaBoostClassifier() :
Recall-> 0.6256212243466953
Precision-> 0.7530294258232064
F1-> 0.6824371455092474

GradientBoostingClassifier() :
Recall-> 0.6926320988351142
Precision-> 0.8364289393036568
F1-> 0.7571621764560994

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              eval_metric='logloss', gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None) :
Recall-> 0.9309523809523809
Precision-> 0.9406593406593406
F1-> 0.9356321839080459
```

(N.B. i parametri di XGBClassifier sono quelli di default, a differenza degli altri però vengono stampati)



Come detto precedentemente, teniamo più in considerazione il parametro di Recall, e possiamo vedere che XGBClassifier è il migliore rispetto agli altri, notiamo anche che il Random Forest Classifier ha dei risultati molto simili.

Ho deciso quindi di eseguire 10 run del programma per vedere quale dei due fosse il migliore, ed i risultati ottenuti sono i seguenti:

XGBClassifier	RandomForestClassifier
0.9352	0.9298
0.9377	0.9277
0.9339	0.9273
0.9444	0.9340
0.9284	0.9168
0.9202	0.9233
0.9211	0.9061
0.9370	0.9240
0.9121	0.9122
0.9285	0.9296

Dai risultati, ho deciso di prendere in considerazione **XGBClassifier.**

6) Fine tuning

In questa parte ci occupiamo di trovare i migliori hyper-parameters per il modello selezionato, inizialmente eseguo una random search, cercando appunto una combinazione casuale tra i seguenti valori:

- N_estimators -> numero di stimatori [100, 150, 200, 250, 300, 350, 400, 450, 500]
- Max_depth -> massima profondità di un albero [3, 4, 5, 6, 7, 8, 9, 10]
- Gamma -> specifica la riduzione minima di perdita per eseguire lo split [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
- Subsample -> esempi da campionare casualmente in ogni albero [0.5, 0.6, 0.7, 0.8, 0.9, 1]
- Scale_pos_weight -> per dataset sbilanciati, negativi/positivi [2]
- Learning_rate -> influenza sul valore corrente dei pesi [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]

Una volta settata la Random Grid utilizzeremo la funzione RandomizedSearchCV per eseguire la random search, e nei parametri della funzione metteremo:

- Estimator -> XGBClassifier
- Param_distribution -> random grid istanziata precedentemente
- Cv -> 10 (cross validation con 10 split)
- Return_train_score -> true (per informazioni dettagliate)
- N_jobs = -1 (esecuzione in parallelo)
- Scoring -> Recall

A questo punto eseguo il fit sul set di dati in training, ed infine stampo i migliori hyper parameters trovati, un esempio:

```
RANDOM SEARCH...
BEST PARAMETERS AFTER RANDOM SEARCH:
Best Score:  0.7637509335324869
Best Hyperparameters:
 {'subsample': 0.9, 'scale_pos_weight': 2, 'n_estimators': 300, 'max_depth': 7, 'learning_rate': 0.01, 'gamma': 0.4}
```

Dopo la random search, e dopo aver trovato i migliori hyper parameters, ho eseguito un Grid Search, però eseguita solamente sul parametro di regolarizzazione reg_alpha, che è un termine di regolarizzazione L1 sui pesi, più aumenta e più il modello sarà conservativo.

Ovviamente la grid search non verrà eseguita sui parametri di default di XGBClassifier, ma sui migliori parametri trovati dalla random search.

```
xgb = XGBClassifier(eval_metric='logloss')

#set the parameters of the random search to the model
xgb.set_params(**best_params)
```

I valori di reg_alpha sui quali faremo la grid search sono 0.00001, 0.001, 0.1, 1, 100, 200.

Una volta settati i valori di reg_alpha, chiamo la funzione GridSearchCV per eseguire la grid search, e nei parametri della funzione metto:

- Xgb -> il modello
- Value_reg_alpha -> i valori sopra citati di reg_alpha
- N_jobs = -1 -> per esecuzione in parallelo
- Cv = 10 -> cross validation con 10 split
- Scoring = Recall
- Return_train_score = true

Eseguo il fit sul set di dati in training, ed infine stampo il miglior risultato ottenuto con il valore di reg_alpha, un esempio:

```
GRID SEARCH...
BEST PARAMETERS FOR REG_ALPHA AFTER GRID SEARCH:
Best Score:  0.7887696041822256
Best Hyperparameters:  {'reg_alpha': 100}
```

7) Final evaluation

In questa fase finale nella variabile best_params sono contenuti i migliori parametri ottenuti dalla random search ed il miglior parametro di reg_aplha ottenuto con la grid search.

Creo il modello XGBClassifier e setto i parametri come i best_params trovati.

Eseguo il fit con il set di dati di training.

Infine, eseguo una predizione sui dati di test, per vedere come si comporta il modello si un set di dati mai visto prima d'ora, per vedere se riesce o meno a generalizzare.

A questo punto stampo i valori di Recall, Precision ed F-1, un esempio:

```
TEST
RECALL -> 0.9952830188679245
PRECISION -> 0.31753197893152746
F1 -> 0.48146035367940676
```

Come precedentemente detto, teniamo molto più in considerazione il parametro di **recall**, per i motivi sopra citati.

Notiamo un incremento di questo valore rispetto al modello con i parametri di default, questo ci fa capire che la parte di fine-tuning in questo esempio è servita molto al modello per permettergli sia di generalizzare sia di comprendere meglio i dati.