**Report sulla classificazione dei livelli di rischio**

1. **Introduzione**

Questo breve report ha lo scopo di spiegare quali sono gli obbiettivi di questa prima fase di studio, come configurare il processo di validazione e quali sono gli algoritmi utilizzati al suo interno.

1. **Obbiettivi**

Sono stati due gli obbiettivi principali per questa prima parte di analisi:

1. Creare un file .pdf che contiene l’albero di decisione utilizzato dall’algoritmo *Decision Tree*, al fine di capire quali sono i fattori che discriminano una classe dall’altra;
2. Realizzazione di uno script python che permette la classificazione di quelli che da Reale Mutua vengono individuati come *livelli di rischio.*
3. **Configurazione**

Sarà necessario settare i seguenti parametri nel file MainApp.py per eseguire lo script:

* *dataset\_path*: path del dataset per cui si vuol effettuare la cross validation
* *n\_fold*: numero di fold utilizzati dalla cross validation
* *risk\_levels*: lista dei livelli di rischio per cui si vuol effettuare la cross validation

1. **Scripting**

Il codice è organizzato in 4 diversi file presenti nella directory *Run*:

* *MainApp.py*: rappresenta il core del codice. Contiene tutte le chiamate a metodi per permettere il corretto processo di validazione;
* *SKLearn.py:* contiene i metodi utili per effettuare la cross validation;
* *ReadDataset*: contiene i metodi per la lettura del dataset di Reale Mutua;
* *txt2xls*: permette la creazione di un file .xls riassuntivo dell’esito della cross validation.

Una seconda directory chiamata *Testing* è fornita all’interno del file zip. Essa sfrutta lo stesso codice presente nella directory *Run,* però i file al suo interno sono programmati per testare tutte le scelte possibili nella configurazione dei parametri e dei classificatori utilizzabili.

* 1. **Algoritmi utilizzati ed organizzazione della logica**

L’idea di base è combinare l’esito di diversi classificatori per effettuare la classificazione.

* + 1. **Lettura del dataset**

Il dataset viene letto grazie al metodo *ReadDataset.read\_csv().*

Esso restituisce l’header, il dataset e il target corrispondenti al dataset per cui si sta effettuando la classificazione.

È importante notare che dal dataset vengono rimossi tutti i quattro livelli di rischio, restituendo solamente quello scelto come target (la scelta avviene tramite passaggio di parametro al metodo).

* + 1. **Label Encoding**

Gli algoritmi di classificazione forniti da scikit-learn consentono di lavorare solamente con valori numerici.

Per tal motivo è necessario un’associazione numero-stringa, la quale viene effettuata dal metodo *SKLearn.LabelEncoder().*

Essa restituisce un dizionario contenete tali associazioni, cosi da poter facilmente passare dal valore numerico alla stringa corrispondente.

Ho scelto l’utilizzo di un Label Encoding creato ad hoc per questo dataset.

Tale scelta è dettata dal label encoder fornito da scikit-learn, il quale riordina tutte le stringhe secondo un ordine alfabetico.

Algoritmi di classificazione come *Decision Tre*e possono soffrire di tale ordinamento, in quanto si basano su criteri di split come x<=y e x>y.

Immaginiamo un encoding della lista [Ancona, Informatica, Palermo] = [1, 2, 3].

La decisione 2<=3 non avrebbe senso logico, difatti il confronto si vorrebbe tra due città e non tra città e professione.

* + 1. **Decision Tree**

All’interno della directory *dtree\_yyyy* (dove *yyyy* rappresenta l’anno del dataset in esame), sono presenti dei file .pdf in cui son presenti gli alberi di decisione associati ad ogni livello di rischio.

Essi servono per capire come lavora l’algoritmo e quali attributi esso considera più discriminanti rispetto agli altri.

* + 1. **Cross Validation**

Il metodo *SKLearn.cross\_validation ()* effettua la Stratified Cross Validation sul dataset in esame, al fine di valutare il funzionamento del processo di classificazione.

Diversi step sono stati fatti che hanno portato alla corrente implementazione del codice:

1. Il primo passo è stato effettuare una semplice classificazione sul dataset, applicando diversi classificatori e configurazioni.

Tale semplicità non ha portato però a buoni risultati, specialmente per il livello di rischio ‘*cost\_caused\_claim*’, che è associato a 4 classi differenti.

Guardando però al significato delle 4 classi (range1, range2, range3 e range4), si è pensato di trasformare il problema in binario, ovvero raggruppare range1, range2 e range3 e separare range4.

Questa scelta è dettata dall’interesse di predire con elevata precisione gli incidenti che causano un grande danno economico per la compagnia assicurativa (range4) piuttosto che gli incidenti meno onerosi.

Nel file *Evolutiva statistiche*/*stat\_all\_clfs\_no\_bag\_no\_over\_no\_cluster.xls* sono presenti i risultati dei sette algoritmi di classificazione testati.

1. Dato il caso di studio, ovvero il lavorare con delle polizze assicurative, è normale avere un dataset non bilanciato, cioè avere la maggior parte delle polizze che non hanno avuto incidente.

Una soluzione a questa problematica è l’utilizzo dell’oversampling e dell’operatore di Bagging.

In particolare, l’oversampling è utilizzato in due versioni (immaginiamo di dover classificare una entry con la classe A o B, dove B è la classe meno popolosa):

* Se il parametro *percentage* del metodo *SKLearn.cross\_validation ()* non è settato, di default l’oversampling lavora per avere un egual numero di elementi appartenenti alla classe A e B nel training set
* Altrimenti, è possibile settare una percentuale per l’oversampling. Ciò significa che è possibile avere un numero di elementi della classe B pari al X% rispetto alla dimensione del dataset di partenza.

Le entry con cui viene fatto oversampling vengono presi in modo casuale dal training set.

Nel file *Evolutiva statistiche*/*stat\_all\_clfs\_yes\_bag\_yes\_over\_no\_cluster.xls* sono presenti i risultati dei sette algoritmi di classificazione testati, con l’applicazione dell’operatore di Bagging ed Oversampling.

1. Dopo aver visionato i risultati del processo spiegato precedentemente (analizzando in particolare le matrici di confusione relative ai vari tentativi di classificazione) è stato notato che il richiamo associato alla classe meno popolosa è sempre piuttosto alto (circa 75%).

Per raffinare il processo di classificazione sono stati messi insieme più classificatori, ovvero:

* Vengono ordinati diversi classificatori in base al valore del richiamo (dal più alto al più basso)
* Eseguo la classificazione
* Interseco i risultati dei diversi algoritmi, per trovare quanti classificatori hanno predetto un’entry come classe B
* Se trovo un numero di classificatori X > n\_right (dove n\_right è un parametro a scelta dell’utente) che hanno predetto tale entry come classe B, allora assegnerò a tale entry classe B, altrimenti classe A.

Inoltre, è stata apportata una modifica al metodo con cui si fa oversampling.

Piuttosto che applicare un oversampling “randomico”, ovvero andando a campionare in modo casuale gli elementi dal dataset di training fino a raggiungere la percentuale desiderata, è stato utilizzato un algoritmo di clustering al fine di avere nel dataset di training finale un egual numero di elementi appartenenti ad ogni cluster.

Nel file *Evolutiva statistiche*/*stat\_all\_clfs\_yes\_bag\_yes\_over\_no\_cluster.xls* sono presenti i risultati delle varie combinazioni testate, con l’applicazione dell’operatore di Bagging ed Oversampling con clustering.

Utilizzando tutte le strategie sopra citate, sono stati notati dei miglioramenti nella predizione per alcuni livelli di rischio (vedere il file *Evolutiva statistiche*/*summary.xls* per vedere l’andamento di precisione e richiamo nel tempo).

*NCD e n\_caused\_claim* sono i livelli di rischio in cui si sono riscontrati i risultati migliori, ovvero F1-Score tra il 40% e il 45%.

*NNC e cost\_caused\_claim* sembrano invece essere i livelli di rischio più difficili da predire, per la quale si raggiunge un F1-Score rispettivamente del 13% e del 20%.

1. **Visione dei risultati della Cross Validation**

I risultati vengono automaticamente generati, a seguito dell’esecuzione degli script, nelle cartelle *Run*/*statistics*\_ *yyyy* e *Testing*/*statistics*\_ *yyyy* (dove *yyyy* rappresenta l’anno del dataset in esame).

Vengono generati due tipi di file:

* *Statistics\_risk\_level\_yyyy.txt*: file .txt in cui sono salvate la matrice di confusione, richiamo, precisione e f1-score relativo ad ogni esecuzione dello script.

Vengono indicati anche i parametri (come percentuale di oversampling, numero di classificatori utilizzati...) con cui è stato fatto lanciato lo script;

* *Statistics\_yyyy.xls*: file .xls che racchiude tutte le statistiche relative a quell’anno cosi da essere più leggibili. Ogni pagina del file Excel è relativa ad un livello di rischio.

Nella directory *Evolutiva statistiche* sono stati raccolti già i risultati dei vari step.

Poiché siamo interessati a vedere quali sono stati i progressi della classificazione, partendo dall’utilizzo di un solo classificatore fino all’utilizzo combinato di vari classificatori, è presente all’interno della directory *Evolutiva statistiche* il file *summary.xls* in cui viene plottato l’avanzamento di Precision, Recall e F1-Score per la classe meno popolosa.

È stata omessa la classe più popolosa in quanto si raggiungono sempre risultati tra il 98% e il 100% (data la numerosità di entry).

1. **Testing**

La directory *Testing* contiene una versione degli script sopra citati che permette il testing delle varie configurazioni in maniera del tutto automatica.

Le varie configurazioni comprendono variazioni dei parametri *n\_right, n\_of\_clf* (numero di classificatori utilizzati) e *percentage* (percentuale sull’oversampling).