German Credit Data
Report per l'Esame di Machine Learning

Luca Inghilterra

N°142715
UNIMORE - Ingegneria Informatica - sede di Mantova
271539@studenti.unimore.it

Abstract

All'interno del sistema bancario, quando una filiale qualsiasi riceve una richiesta di prestito è sempre necessario vagliarla con scrupolosità, in base al profilo del richiedente. Due tipi di rischi sono infatti associati alla decisione della banca:

- se il richiedente è un buon rischio di credito il rifiuto del prestito si traduce in una perdita di affari;
- se il richiedente è un cattivo rischio di credito l'approvazione del prestito comporta una perdita finanziaria.

Questo progetto si pone come obiettivo la creazione di uno strumento di predizione per stimare la solvibilità del richiedente, sfruttando la classificazione binaria.

1 Introduzione

Il progetto è stato sviluppato in codice python sfruttando i metodi delle librerie `numpy`, `pandas`, `matplotlib`, `seaborn`, `warning`, `sklearn`, `time` e `colorama`. Per maggiore chiarezza si è anche deciso di suddividere i passi di sviluppo seguiti all'interno di moduli e funzioni, omonimi alle seguenti sezioni e sottosezioni. Inoltre, nell'area "OPTIONS" all'inizio del programma è presente la costante PLOT, settabile a `True` o `False` rispettivamente per attivare o disattivare la creazione di tutti i grafici.

2 Data Acquisition

Per lo svolgimento di questo progetto è stato scelto il dataset "German Credit Data". In esso sono presenti i dati registrati dei richiedenti per un prestito ad un ipotetico istituto bancario tedesco. Sono disponibili 1000 samples e 20 features, delle quali sono:

- 7 numeriche: `"Duration in month"`, `"Credit amount"`, `"Installment rate in percentage of disposable income" "Present residence since"`, `"Age in years"`, `"Number of existing credits at this bank"`, `"Number of people being liable to provide maintenance for"` ;
- 13 categoriche: `"Status of existing checking account"`, `"Credit history"`, `"Purpose"`, `"Savings account/bonds"`, `"Present employment since"`, `"Personal status and sex"`, `"Other debtors / guarantors"`, `"Property"`, `"Other installment plans"`, `"Housing"`, `"Job"`, `"Telephone"`, `"Foreign worker"` .

Il dataset, essendo supervisionato, è fornito di una ground truth numerica `"y"`. I valori che essa può assumere sono:

- 1: il richiedente è solvibile;
- 2: il richiedente non è solvibile.

Date le sue caratteristiche il dataset si presta bene ad essere utilizzato all'interno di un modello di classificazione binaria e quindi alla stima di solvibilità che si vuole attuare.

2.1 Dataset Splitting

Disponendo di un dataset unico, durante lo sviluppo del progetto si è reso necessario eseguire la sua suddivisione in "training set" e "testing set". Perciò, per simulare nel programma di avere effettivamente due dataset distinti si è deciso di gestire l'operazione all'interno della funzione "Data Acquisition".
Secondo il criterio di splitting seguito si è dedicato l'80% dei sample alla costituzione del training, mentre il rimanente 20% è stato destinato al testing. Inoltre:

- per favorire l'ottenimento di risultati ripetibili, considerando lo shuffle a cui sono sottoposti i sample prima di essere suddivisi, si è scelto di fornire un seed alla funzione di splitting `train_test_split()`;
- per contrastare lo sbilanciamento delle classi nel dataset di partenza, durante lo splitting è stata impostata nella stessa `train_test_split()` la stratificazione dei samples tra i due set in funzione della colonna della ground truth.

3 Exploratory Data Analysis

Preso il training set, questa fase si è suddivisa nelle seguenti sottosezioni.

3.1 Exploratory Data Analysis with raw data

Prima di adoperare qualunque modifica sui dati, in questa fase:

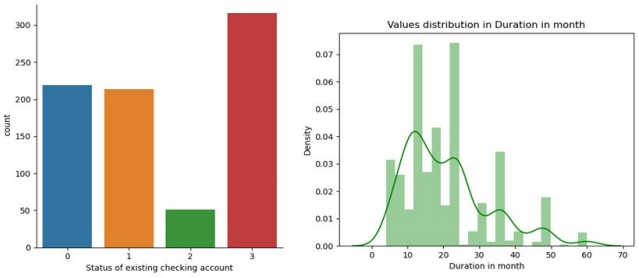
- si è stampato a schermo le informazioni generali sul training set e alcuni dei sample che lo compongono, per comprenderne meglio la conformazione;
- si è suddiviso il training set nella design matrix $X_{training}$ e nel target vector $y_{training}$;

3.2 Features Preprocessing

Dopo l'EDA iniziale si arriva a questa funzione, nella quale:

- il training set è stato controllato per evitare la presenza di valori nulli, spazi o samples duplicati;
- le features categoriche sono state raggruppate e codificate con valori discreti, in modo tale da poter essere utilizzabili con i modelli predittivi forniti da `sklearn`.

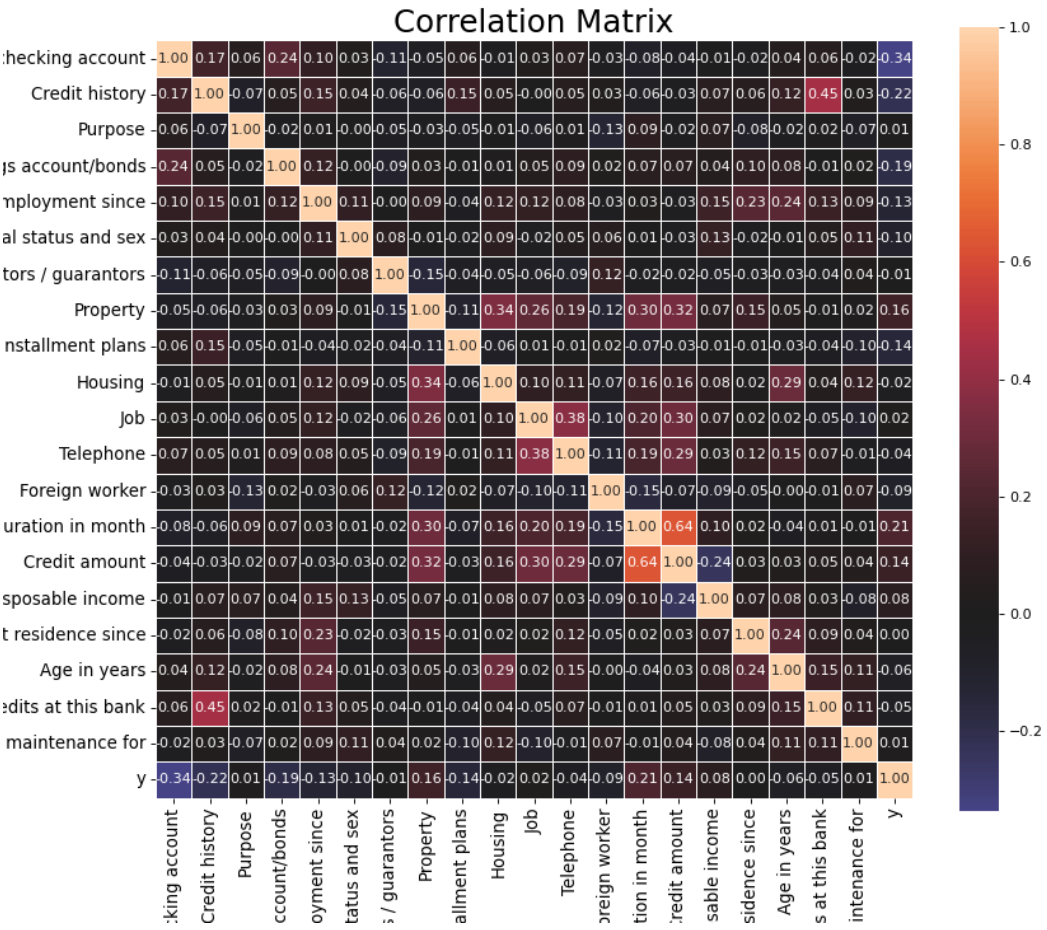
Inoltre, per dare una maggiore comprensibilità, sono stati aggiunti dei `catplot` (destra) e dei `distplot` (sinistra) per graficare le rispettive distribuzioni dei valori delle feature categoriche trasformate in discrete e numeriche.



La correlazioni maggiore presente tra ground truth e feature si verifica con anche il `catplot` per confronto tra la distribuzione dei valori tra la correlation matrix.

3.3 Exploratory Data Analysis with clean data

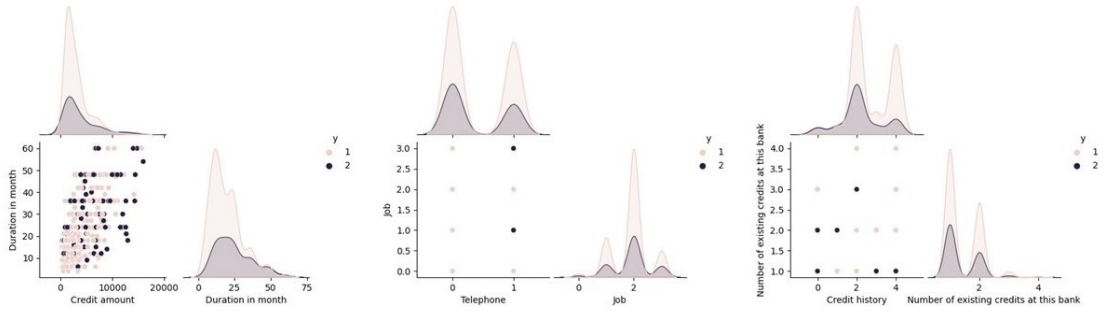
In questa EDA secondaria, nella quale si trattano i dati dopo le modiche apportate durante il primo preprocessing, si è calcolata la correlation matrix per vagliare qualunque correlazione tra le features stesse o tra le features e la ground truth.



Dal correlogramma si evince come le feature più correlate a coppie siano:

- `Credit amount` con `Duration in month`;
- `Telephone` con `Job`.
- `Credit at this bank` con `Credit history`;

Ecco qui di seguito il confronto delle tre coppie nei `pairplot`:



3.4 Numerical Features Scaling

In questa sottosezione, come conclusione dell'EDA, i valori delle features numeriche vengono normalizzati secondo la Distribuzione Gaussiana Standardizzata. Tale operazione si rende utile in quanto alcuni dei modelli utilizzati, basandosi sul Gradient Descent o Ascent, potrebbero avere degli aggiornamenti disomogenei tra le componenti del vettore dei parametri θ.

4 Models Selection

Tra i modelli disponibili per costruire il classificatore binario sono stati scelti:

- Logistic Regression, in quanto molto efficiente, badandosi su un modello parametrico lineare, e facile da addestrare, seppur necessiti dello scalamento delle feature numeriche, essendo basata sul Gradient Descent o Ascent;
- K-Nearest Neighbors, poiché semplice e facile da implementare e non necessiti dell' ottimizzazione parametri, ed anche perché in dataset di piccola dimensione come quello desiderato non risulti eccessivamente oneroso;
- Decision Tree, essendo poco costoso da costruire, estremamente veloce ad inference time ed inoltre comodo da utilizzare nel caso di un piccolo dataset come quello considerato;
- Support Vector Classifier, che seppur nel nostro caso sia un po' più lento rispetto ai classificatori citati precedentemente, offre una buona resistenza all'overfitting.

4.1 Grid Search

All'interno di questa fase si è utilizzato il "validation set" per ottenere i migliori iperparametri con cui settare ciascun modello. Tali iperparametri sono memorizzati all'interno di una griglia di possibilità e sono ottenuti basandosi sull'esperienza ottenuta con prove ripetute.
Dunque, all'interno del progetto si è cercato, seguendo la convergenza vista in ciascuna prova eseguita, di restringere in larghezza ed estendere in profondità l'intorno dei valori utilizzato, in modo da minimizzare la distanza tra il valore ideale e quello a disposizione.
In particolare, il validation set qui utilizzato è stato calcolato internamente alla classe `GridSearchCV()` mediante una 5-fold Cross Validation stratificata.
Ecco gli iperparametri testati per ogni modello:

- Logistic Regression: **penalty** $\in\{L1,\ L2\}$, **C** $\in\{10^{-5}, 5^{-5}, 1^{-4}, 5^{-4}, 0.01, 0.05, 0.07, 0.08, 0.09, 0.1, 0.5, 1$
- K-Nearest Neighbors: **k** $\in\{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}$;
- Decision Tree: **fit-prior** $\in\{True,\ False\}$;
- Support Vector Classifier: **C** $\in\{10^-4, 10^-2, 1, 10^1, 50, 10^2\}$, **gamma** $\in\{0.005, 0.004, 0.003, 0.002, 0.001, 0.0005\}$;
  Conseguentemente, gli iperparametri migliori ottenuti per ogni modello sono stati:

```
###################### GRID SEARCH ######################

LOGISTIC REGRESSION:
Accuracy:    0.72
    --> best value for hyperparameter "penalty":   l1
    --> best value for hyperparameter "C":   0.09

K-NEAREST NEIGHBORS:
Accuracy:    0.7474999999999999
    --> best value for hyperparameter "n_neighbors":   13

DECISION TREE:
Accuracy:    0.67375
    --> best value for hyperparameter "criterion":   gini

SUPPORT VECTOR CLASSIFIER:
Accuracy:    0.7212500000000001
    --> best value for hyperparameter "C":   100.0
    --> best value for hyperparameter "gamma":   0.004
    --> best value for hyperparameter "kernel":   rbf
```

4.2 Cross Validation

A questo punto del progetto, usufruendo della funzione `cross_validate()` si calcolano le metriche di prestazione per ciascun singolo modello tra quelli scelti, provandoli tutti sul validation set con la 5-fold stratificata.

Model	Accuracy	Weighted F1-score
Logistic Regression	0.72	0.7298057169483806
K-Nearest Neighbors	0.74375	0.7067608485447154
Decision Tree	0.6762499999999999	0.6706515350978982
Support Vector Classifier	0.7212500000000001	0.7287638383211678

Table 1: Metriche di prestazione per i singoli modelli

Considerando il Teorema di Condorcet, nel progetto si è ritenuto utile calcolare le stime di performance anche sui seguenti modelli ensemble:

- Stacking Classifier con Logistic Regression, K-Nearest Neighbors e Support Vector Classifier come weak learner, ovvero i modelli aventi le stime di performance più alte tra quelli scelti;
- Stacking Classifier con Logistic Regression, K-Nearest Neighbors e Decision Tree come weak learner, scelto in queste componenti per confrontare le sue prestazioni senza il modello SVC rispetto al precedente ensemble;
- Bagging Classifier composto da 11 Decision Tree (noto anche come *Random Forest*), che nel progetto si può considerare un interessante rivale dei primi due, basandosi su una logica implementativa differente sfruttando solo il modello peggiore, ma essendo di gran luna più veloce dei primi due.

Model	Accuracy	Weighted F1-score
Stacking - Log, KNN, SVC	0.7537499999999999	0.7382117376118748
Stacking - Log, KNN, DT	0.7350000000000001	0.7138914157556597
Random Forest	0.69625	0.6866234877929683

Table 2: Metriche di prestazione per ciascun ensemble

4.3 Final Model

Dalla tabella appena vista si evince come le stime ottenute dai modelli ensemble siano leggermente migliori rispetto a quelle dei singoli. Inoltre, data la loro natura di modelli compositi, gli ensemble dispongono di una maggiore affidabilità nei risultati, a differenza invece dei modelli singoli, più suscettibili a peggiorare le prestazioni in caso di overfitting per esempio.
In seguito a queste considerazioni, complici i valori delle sue metriche, la scelta più naturale ricadebbre sullo *Stacking - Log, KNN, SVC*. Tuttavia, sarebbe utile considerare un'altra misurazione: il tempo impiegato per la validazione.

Model	Validation period
Stacking - Log, KNN, SVC	1435.5371143817902 [sec]
Stacking - Log, KNN, DT	48.60026788711548 [sec]
Random Forest	0.17180204391479492 [sec]

Table 3: Periodo di validation per ciascun ensemble

Dopo aver visto quest'ultima tabella risulta evidente che, per gli scopi perseguiti in questo progetto, la miglior soluzione consista nel compromesso tra tempistiche ed prestazioni, quindi nello *Stacking - Log, KNN, DT*.

5 Feature Selection

All'interno del progetto è implementata e funzionante la feature selection sul modello finale secondo la classe `SequentialFeatureSelector()`. Tuttavia, a seguito di numerosi test, si è notato come essa sia una componente "time consuming" del programma, impiegando 362,45 [sec]. Per tale ragione è stata presa la decisione di non utilizzarla, mantenendola commentata.

6 Training

Giunti a questa funzione, il modello finale scelto, lo Stacking - Log, KNN, DT Classifier, viene trainato sull'intero training set in funzione di tutte le features. L'operazione impiega 11.12 [sec].

7 Testing

Finito di trainare il modello, si è giunti alle lavorazioni da svolgere con il testing set.

7.1 Exploratory Data Analysis

Anche con il training set:

- sono state stampate a schermo le sue informazioni generali e alcuni dei sample che lo compongono, per comprenderne meglio la conformazione;
- si è adoperata la suddivisione nella design matrix $X_{testing}$ e nel target vector $y_{testing}$.

7.2 Features Preprocessing

Tornando al preprocessing per il testing set:

- esso è stato controllato per evitare la presenza di valori nulli, spazi o samples duplicati;
- le features categoriche sono state raggruppate e codificate con valori discreti, in modo tale da poter essere utilizzabili con i modelli predittivi forniti da `sklearn`.

7.3 Numerical Features Scaling

Continuando la preparazione del testing i valori delle features numeriche vengono normalizzati secondo la medesima distribuzione usata per il training. Tale operazione si rende utile in quanto alcuni dei modelli utilizzati, basandosi sul Gradient Descent o Ascent, potrebbero avere degli aggiornamenti disomogenei tra le componenti del vettore dei parametri θ.

7.4 Prediction

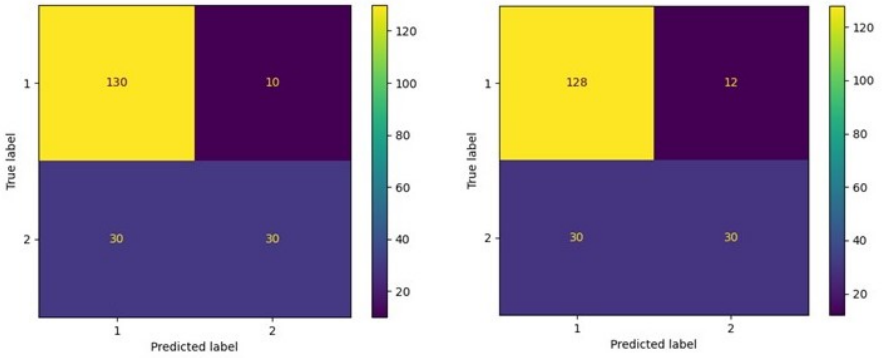
In questa fase si è ottenuta la predizione su $X_{testing}$, trovando $y_{prediction}$.

8 Final Testing Results

Una volta ottenuto $y_{prediction}$ esso è stato confrontato con $y_{testing}$, ottenendo i risultati della tabella 4.
Tra le varie prove durante lo sviluppo del progetto si è giunti al testing finale senza feature selection anche con lo *Stacking - Log, KNN, SVC*. Dunque, da come si nota nel confronto delle confusion matrix alla pagina seguente, lo *Stacking - Log, KNN, SVC* (sinistra) e lo *Stacking - Log, KNN, DT* (destra) hanno raggiunto risultati quasi coincidenti nel determinare TP, TN, FP e FN.
Questo risultato ha quindi giustificato la scelta di un modello leggermente meno performante a vantaggio dell'efficienza del progetto.

Model	Accuracy	Precision	Recall	F1-score
Stacking - Log, KNN, DT	0.79	0.781374321880651	0.79	0.7778128701144886

Table 4: Metriche di prestazione finali



Per concludere si ottiene il grafico della curva ROC ed il suo valore dell'integrale AUC: