Report

Descrizione

Il dataset analizzato contiene dati riguardo le specifiche di CPU e GPU rilasciate sul mercato nel corso degli anni.

Obiettivo

Date le caratteristiche numeriche di un nuovo processore, determinarne il produttore.

Dataset

Il dataset produttore-nuovo-processore è composto da 4854 righe e 13 colonne:

- **Product: il prodotto**
- **Type: la tipologia del prodotto (CPU, GPU)**
- **Release Date: data di rilascio del prodotto**
- **Process Size (nm): dimensione dei transistor**
- **TDP (W): consumo del processore**
- **Die Size (mm^2): dimensione del processore**
- **Transistors (million): numero di transistor nel processore**
- **Freq (MHz): frequenza del processore**
- **Foundry: da chi è stato prodotto**
- **Vendor: da chi è commercializzato**
- **FP16 GFLOPS: miliardi di operazioni in virgola mobile per secondo su 16 bit**
- **FP32 GFLOPS: miliardi di operazioni in virgola mobile per secondo su 32 bit**
- **FP64 GFLOPS: miliardi di operazioni in virgola mobile per secondo su 64 bit**

Operazioni sul Dataset

Prima di procedere con l'analisi esplorativa dei dati, ho apportato delle modifiche al dataset sia in Python che R.

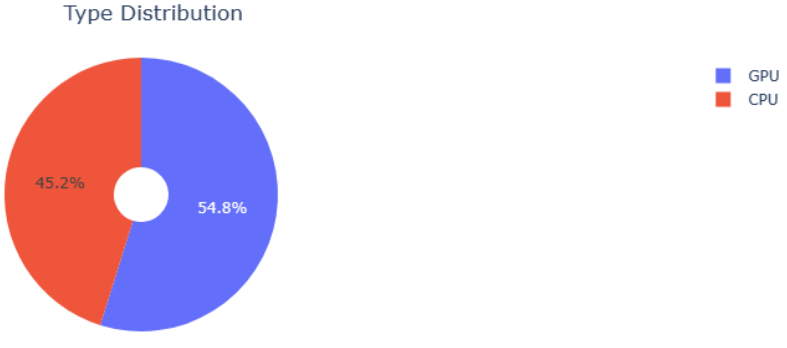
Ho convertito la colonna 'Release Date' in un formato di data e ora e ho creato una nuova colonna chiamata 'Year', assegnando ad essa il valore dell'anno estratto dalla colonna 'Release Date'. Ho deciso di utilizzare l'anno come una caratteristica numerica supplementare.

Ho sostituito i valori con una frequenza inferiore a 100 nella colonna 'Foundry' con il valore 'Other'. Le nuove classi sono 5: Unknown, Intel, GF, TSMC, Other. Unknown rappresenta un produttore non identificato, potrebbe essere diverso da quelli elencati nel dataset o anche uno tra essi. In mancanza di informazioni certe, ho deciso di lasciarlo invariato.

Ho riempito i valori mancanti (NA) in colonne specifiche del DataFrame con la media dei valori presenti in quella colonna.

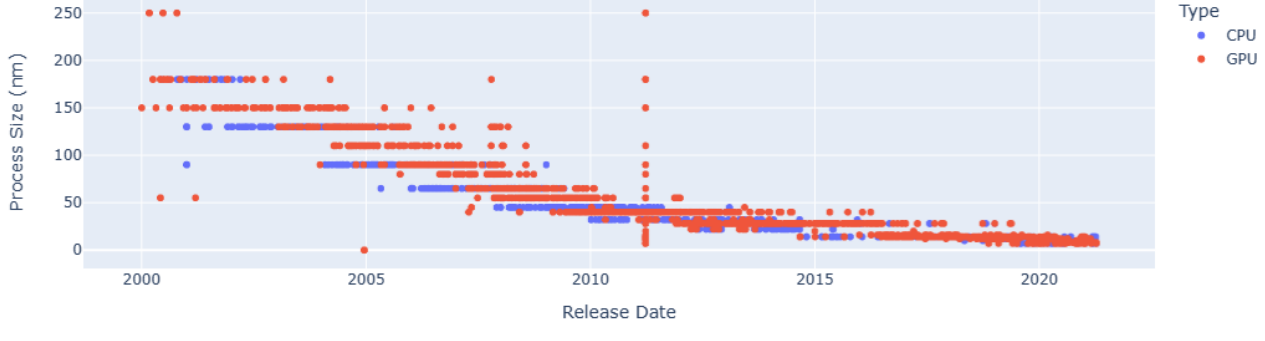
EDA (Exploratory Data Analysis)

A seguire ho creato una serie di grafici per analizzare in profondità il dataset e acquisire ulteriori informazioni.

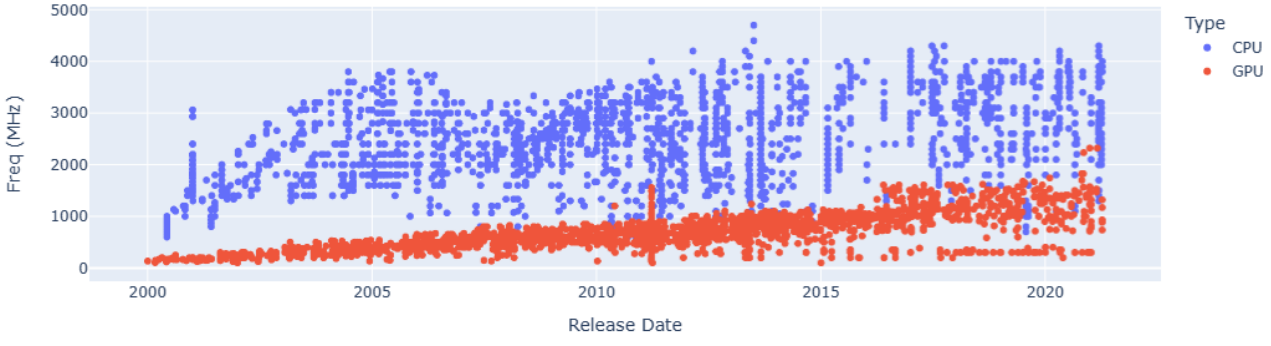


Il primo è un grafico a ciambella che rappresenta la distribuzione dei valori della colonna 'Type'. Il 54,8% dei valori totali sono GPU e il resto sono CPU.

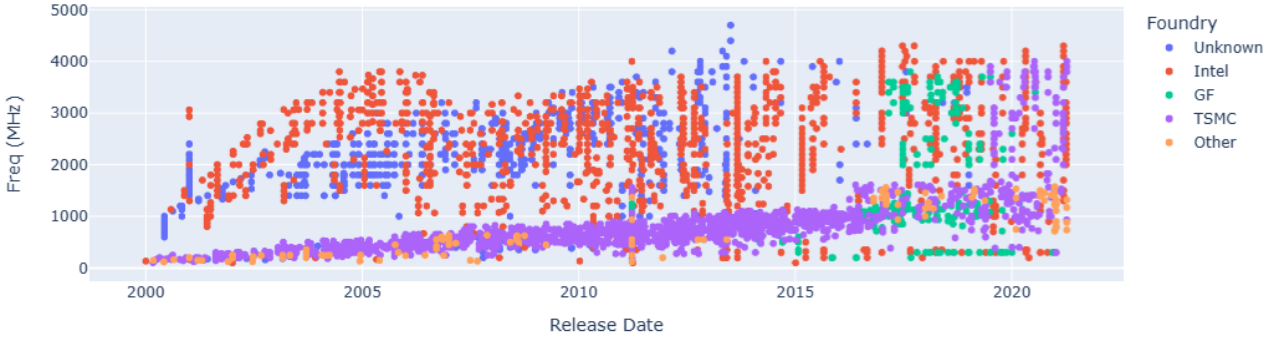
Successivamente ho creato dei grafici a dispersione. Di seguito riporto i più rilevanti, ovvero quelli che sono visivamente più efficaci per ottenere informazioni.



Si può chiaramente osservare che sia per la CPU che per la GPU, la dimensione del transistor è in continua diminuzione.

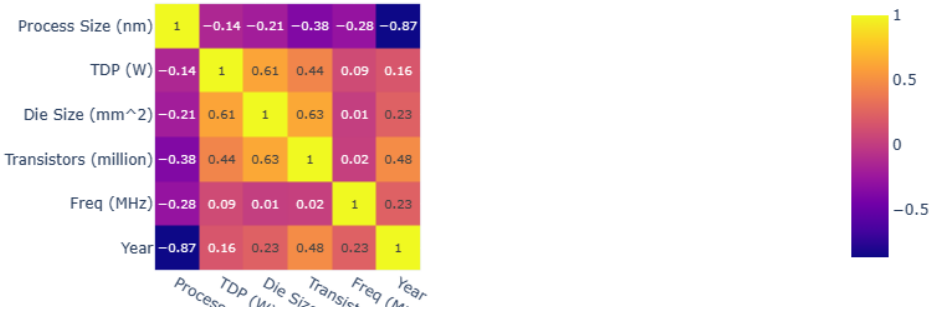


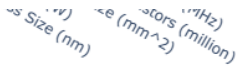
E' evidente che la frequenza dei processori CPU è sempre maggiore rispetto a quella dei processori GPU.



Notiamo come GF sia entrato recentemente sul mercato e negli ultimi anni abbia raggiunto valori di frequenza simili a quelli di Intel e TSMC. Sebbene TSMC sia presente da molto tempo, i suoi valori di frequenza sono aumentati di recente. Intel appare invece molto coerente nei suoi valori di frequenza.

Poi ho separato le features dalla variabile target, poiché desideravo creare un correlogramma esclusivamente su quelle caratteristiche numeriche che ho utilizzato nei successivi modelli.



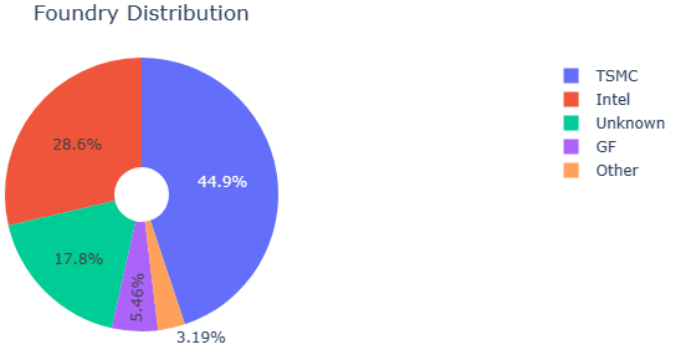


Si può osservare come la correlazione positiva più significativa sia quella tra le variabili 'Die Size' e 'TDP' e tra 'Transistor' e 'Die Size'. Con l'aumento della dimensione del processore, aumenta anche il consumo del processore e il numero di transistor, e viceversa. La correlazione negativa più significativa è quella tra le variabili 'Year' e 'Process Size'. Come già evidenziato dal grafico precedente, con il passare degli anni la dimensione del transistor è diminuita.

Il Dataset è Bilanciato?

Ho verificato se il dataset fosse bilanciato, ovvero se contenesse un numero uguale di istanze per ogni classe.

In caso di sbilanciamento significativo, sarebbe necessario effettuare operazioni di ribilanciamento. Questo fenomeno può causare notevoli difficoltà nella classificazione, poiché tende ad assegnare le istanze alla classe più rappresentata.

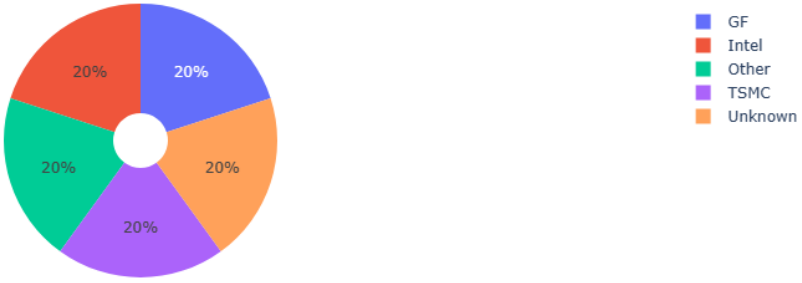


Come si può evincere dal grafico, il dataset presenta un forte sbilanciamento. Ho utilizzato tecniche di ribilanciamento sia in Python che in R.

Approccio in Python

Prima di applicare qualsiasi tecnica, ho scalato i dati e ho suddiviso il dataset in set di addestramento e di test poiché utilizzare lo stesso dataset comporta il rischio di non valutare la capacità dell'algoritmo di generalizzare. Il dataset di addestramento è molto più ampio rispetto a quello di test (75% contro il 25% del dataset originale).

A questo punto ho utilizzato un metodo di sottocampionamento per ridurre la presenza delle classi maggioritarie. Una tecnica di sottocampionamento basata sul mantenimento delle istanza, chiamata Near-Miss. In particolare ho utilizzato la variante di tipo 1, in cui vengono mantenute le istanze della classe di maggioranza che hanno la minima distanza media dalle tre istanze più vicine della classe di minoranza.



Dal grafico è possibile osservare che il dataset (training) è finalmente bilanciato, poiché ogni classe presenta 108 istanze.

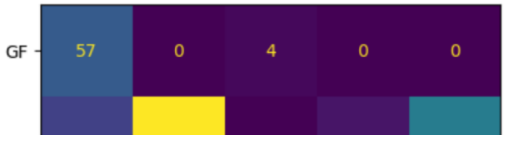
Dopodiché ho utilizzato la Cross-validation per valutare la performance di alcuni modelli di apprendimento automatico sul dataset di addestramento. Il suo scopo è quello di fornire una stima più affidabile della performance del modello rispetto alla semplice valutazione su un singolo set di dati di test.

Per valutare la performance dei diversi algoritmi di classificazione, ho utilizzato una suddivisione in 5 gruppi dei dati di addestramento invece del valore tipico di 10 gruppi. Ho testato 5 algoritmi di classificazione diversi: Alberi di Decisione, Naive Bayes Gaussiano, Random Forest, Support Vector Classification (SVC) e Regressione Logistica. Tutti gli algoritmi sono stati utilizzati con i loro parametri predefiniti.

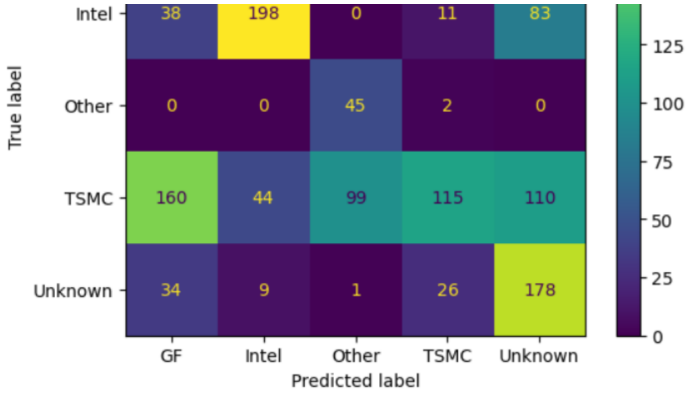
```
Cross-validation scores for DecisionTreeClassifier: [0.68518519 0.73148148 0.78703704 0.84259259 0.73148148]
Mean cross-validation score for DecisionTreeClassifier: 0.76

Cross-validation scores for GaussianNB: [0.55555556 0.52777778 0.41666667 0.57407407 0.60185185]
Mean cross-validation score for GaussianNB: 0.54

Cross-validation scores for RandomForestClassifier: [0.66666667 0.82407407 0.82407407 0.91666667 0.78703704]
Mean cross-validation score for RandomForestClassifier: 0.80

Cross-validation scores for SVC: [0.59259259 0.67592593 0.59259259 0.62037037 0.56481481]
Mean cross-validation score for SVC: 0.61

Cross-validation scores for LogisticRegression: [0.5        0.51851852 0.44444444 0.5        0.4537037 ]
Mean cross-validation score for LogisticRegression: 0.48
```

Dopo aver eseguito la cross-validation sui dati di addestramento, ho scoperto che l'algoritmo di classificazione LogisticRegression() ha ottenuto il punteggio medio più basso mentre RandomForestClassifier() ha ottenuto il punteggio medio più alto.

Successivamente ho creato una funzione di classificazione per poter applicare l'algoritmo migliore ai miei dati e valutare le sue prestazioni. La funzione restituisce informazioni come l'accuratezza sia per il dataset di addestramento che per quello di test, la precisione, il recall, il punteggio F1, un breve report e la matrice di confusione. La funzione è stata progettata in modo da poter facilmente cambiare l'algoritmo di classificazione utilizzato e comparare le prestazioni tra diversi algoritmi. Ho scelto di utilizzare RandomForestClassifier() per la classificazione dei miei dati.




```
Random Forest
Training Accuracy: 0.9963
Test Accuracy: 0.4885
Precision: 0.5029
Recall: 0.6855
```

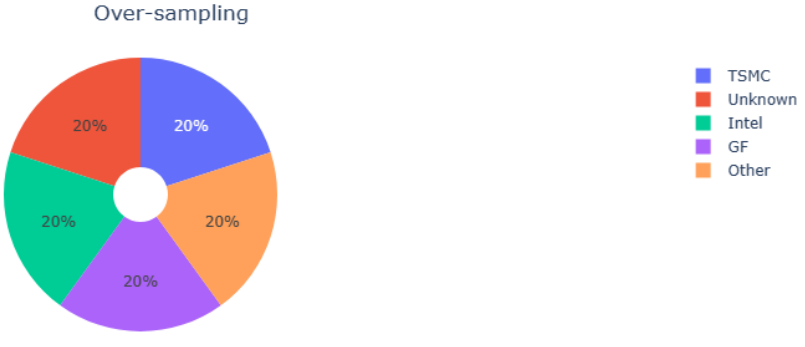



```
F1 Score: 0.4758

              precision    recall  f1-score   support

          GF       0.20      0.93      0.33        61
       Intel       0.79      0.60      0.68       330
       Other       0.30      0.96      0.46        47
        TSMC       0.75      0.22      0.34       528
     Unknown       0.48      0.72      0.58       248

    accuracy                           0.49      1214
   macro avg       0.50      0.69      0.48      1214
weighted avg       0.66      0.49      0.48      1214
```

Il classificatore Random Forest descritto sembra avere un'alta accuratezza di addestramento (0.9963) ma un'accuratezza di test più bassa (0.4885). Ciò potrebbe indicare un problema di overfitting, cioè il modello è troppo adattato ai dati di addestramento e non generalizza bene alle nuove osservazioni.

La precisione del classificatore è 0.5029, il che significa che circa il 50% delle osservazioni classificate come positive sono effettivamente positive. Il recall è 0.6855, il che significa che il modello individua circa il 68% delle osservazioni positive. Il punteggio F1 di 0.4758 indica che il modello non riesce a bilanciare la precisione e il recall. In generale, sembra che questo modello di Random Forest non stia generalizzando bene ai dati di test e potrebbe beneficiare di una ulteriore ottimizzazione dei parametri o una diversa tecnica di apprendimento automatico.

Dati i risultati ottenuti con il sottocampionamento, ho deciso di esplorare se fosse possibile migliorare le prestazioni utilizzando una tecnica di sovracampionamento, ovvero aumentando la presenza delle classi minoritarie nel dataset di addestramento. Ho utilizzato la tecnica SMOTE (Synthetic Minority Over-sampling Technique). La tecnica genera nuovi esempi sintetici delle classi minoritarie interpolando tra i punti esistenti delle classi minoritarie, in modo da aumentare la rappresentatività di queste classi nei dati di addestramento.



Il dataset è stato nuovamente bilanciato, con un totale di 1650 istanze per ciascuna classe.

Per valutare la performance dei modelli di apprendimento automatico sul dataset di addestramento, ho utilizzato ancora la metodologia della Cross-validation.

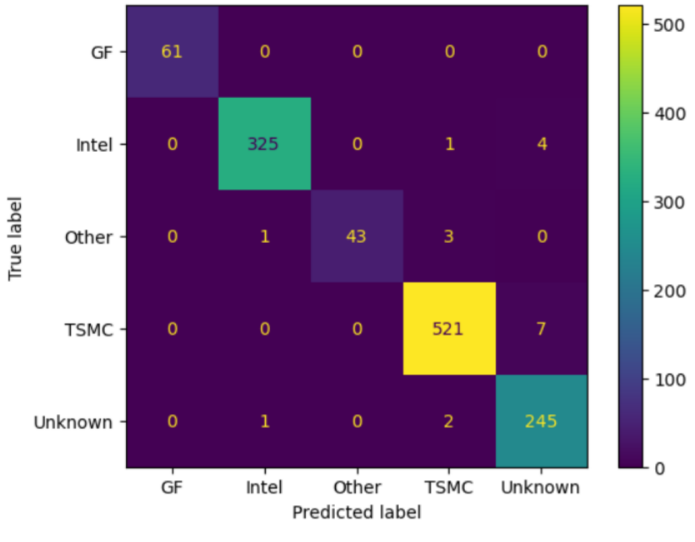
Ho utilizzato la stessa metodologia di suddivisione dei dati di addestramento in 5 gruppi per valutare la performance degli algoritmi di classificazione. Ho testato gli stessi 5 algoritmi utilizzati in precedenza, sempre utilizzati con i loro parametri predefiniti. La scelta di utilizzare lo stesso numero di gruppi e gli stessi algoritmi ci permette di confrontare i risultati con quelli ottenuti precedentemente, e verificare se la tecnica di sovracampionamento abbia migliorato o meno le prestazioni del modello.

```
Cross-validation scores for DecisionTreeClassifier: [0.97090909 0.98424242 0.98363636 0.97575758 0.97393939]
Mean cross-validation score for DecisionTreeClassifier: 0.98

Cross-validation scores for GaussianNB: [0.47393939 0.45030303 0.49515152 0.45757576 0.45636364]
Mean cross-validation score for GaussianNB: 0.47

Cross-validation scores for RandomForestClassifier: [0.99030303 0.99212121 0.99393939 0.98969697 0.99272727]
Mean cross-validation score for RandomForestClassifier: 0.99

Cross-validation scores for SVC: [0.79939394 0.79272727 0.81636364 0.80909091 0.80060606]
Mean cross-validation score for SVC: 0.80

Cross-validation scores for LogisticRegression: [0.61333333 0.62666667 0.65757576 0.63939394 0.62848485]
Mean cross-validation score for LogisticRegression: 0.63
```

Questa volta l'algoritmo di classificazione GaussianNB() ha ottenuto il punteggio medio più basso mentre RandomForestClassifier() ha sempre il punteggio medio più alto.

In generale, i risultati mostrano che la tecnica di sovracampionamento SMOTE ha migliorato le prestazioni dei modelli di apprendimento automatico rispetto all'utilizzo di un semplice sottocampionamento.

In seguito, ho utilizzato la funzione di classificazione creata in precedenza per valutare le prestazioni del modello e ho scelto di utilizzare ancora RandomForestClassifier() per la classificazione dei miei dati.




```
Random Forest
Training Accuracy: 0.9996
Test Accuracy: 0.9843
Precision: 0.9879
Recall: 0.9749
F1 Score: 0.981
```

	precision	recall	f1-score	support
GF	1.00	1.00	1.00	61
Intel	0.99	0.98	0.99	330
Other	1.00	0.91	0.96	47
TSMC	0.99	0.99	0.99	528
Unknown	0.96	0.99	0.97	248
accuracy			0.98	1214
macro avg	0.99	0.97	0.98	1214
weighted avg	0.98	0.98	0.98	1214

Il classificatore Random Forest descritto sembra avere un'alta accuratezza sia per il dataset di addestramento (0.9996) che per quello di test (0.9843). Ciò indica che il modello è in grado di generalizzare bene ai dati di test, e non ci sono segni di

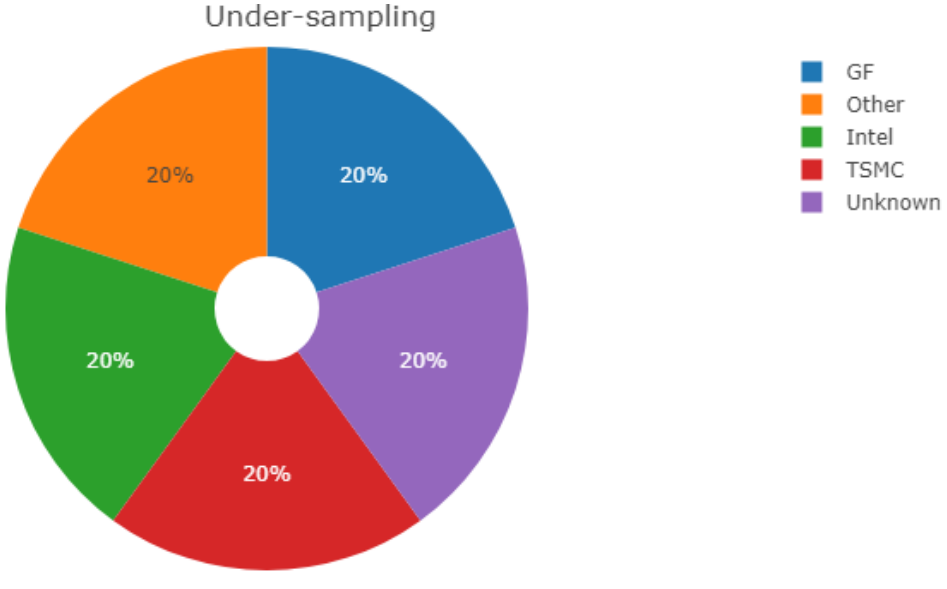
overfitting. La precisione del classificatore è 0.9879, il che significa che circa il 98% delle osservazioni classificate come positive sono effettivamente positive. Il recall è 0.9749, il che significa che il modello individua circa il 97% delle osservazioni positive. Il punteggio F1 di 0.981 indica che il modello riesce a bilanciare bene la precisione e il recall. In generale, questo modello di Random Forest sembra avere ottime prestazioni sia sui dati di addestramento che di test, e potrebbe essere considerato una buona scelta per la classificazione dei dati.

Approccio in R

Le librerie in R sono state caricate una alla volta poiché la funzione della libreria Hmisc cessava di funzionare a causa di un conflitto con un'altra libreria.

Nuovamente, prima di utilizzare qualsiasi metodo, ho eseguito la normalizzazione dei dati e ho suddiviso il dataset in un insieme di addestramento (75%) e uno di test (25%) utilizzando la suddivisione del dataset originale.

In R, a differenza di Python, ho incontrato difficoltà nell'utilizzo di alcune tecniche di sotto/sovracampionamento poiché alcune librerie, come DMwR, non erano disponibili per la mia versione del software. Pertanto, ho optato per un sottocampionamento casuale, nonostante non sia raccomandato, poiché questo metodo non tiene conto della rilevanza delle istanze campionate.



Dal grafico è possibile notare che il dataset di training è stato finalmente bilanciato, poiché ogni classe presenta un numero di istanze pari a quello della classe con il minor numero di istanze.

A questo punto, avrei voluto utilizzare la tecnica di Cross-validation per valutare il mio modello, come ho fatto in Python, ma mi è stato impossibile a causa dei numerosi errori che R mi ha restituito. Ho riscontrato gli stessi problemi anche nell'utilizzo della curva ROC e del punteggio AUC, a causa del fatto che il mio classificatore è multi-

classe. Sarebbe stato molto più semplice utilizzare queste tecniche e altri algoritmi di classificazione su un classificatore binario. Nonostante ciò, ho deciso di utilizzare tre algoritmi di classificazione (Alberi di Decisione, Naive Bayes, Random Forest) per valutare le loro performance e confrontarle tra loro.

Alberi di Decisione:

```
Confusion Matrix and Statistics

          model_dt_under
            GF Other Intel TSMC Unknown
  GF        64     2     1    5       0
  Other      1    26     1   10       3
  Intel      6    14   231   46     117
  TSMC      31    59     0  478      58
  Unknown    5    11    40   12     165

Overall Statistics

               Accuracy : 0.6955
                 95% CI : (0.6705, 0.7197)
    No Information Rate : 0.3975
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5733

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: GF Class: Other Class: Intel Class: TSMC Class: Unknown
Sensitivity            0.59813       0.23214       0.8462       0.8675         0.4810
Specificity            0.99375       0.98823       0.8356       0.8228         0.9348
Pos Pred Value         0.88889       0.63415       0.5580       0.7636         0.7082
Neg Pred Value         0.96728       0.93606       0.9568       0.9039         0.8456
Prevalence             0.07720       0.08081       0.1970       0.3975         0.2475
Detection Rate         0.04618       0.01876       0.1667       0.3449         0.1190
Detection Prevalence   0.05195       0.02958       0.2987       0.4517         0.1681
Balanced Accuracy      0.79594       0.61018       0.8409       0.8451         0.7079
```

In base alle misure di prestazione mostrate, si può dire che il modello di Decision Tree in questione ha un'accuratezza generale del 69,55%, il che significa che il modello classifica correttamente circa il 70% delle osservazioni. Il Kappa è di 0.5733, che è una misura della concordanza tra la classificazione del modello e la classificazione reale, valori più alti indicano una maggiore concordanza.

La sensibilità e la specificità per ogni classe sono anche importanti per valutare la performance del modello. La sensibilità indica la percentuale di osservazioni effettivamente positive che sono state classificate correttamente dal modello, la specificità indica la percentuale di osservazioni effettivamente negative che sono state classificate correttamente dal modello.

In generale, si può dire che il modello ha una performance discreta ma non eccellente.

Naive Bayes:

```
Confusion Matrix and Statistics

          model_nb_under
            GF Other Intel TSMC Unknown
  GF        60     0     5    7       0
  Other     12     0     7    3      19
  Intel    117     1   158   13     125
  TSMC      66    28   216  120     196
  Unknown    2     3    47    0     181

Overall Statistics

               Accuracy : 0.3745
                 95% CI : (0.3489, 0.4005)
    No Information Rate : 0.3759
```

```
    P-Value [Acc > NIR] : 0.5542

                  Kappa : 0.2047

 Mcnemar's Test P-Value : <2e-16

Statistics by Class:

                     Class: GF Class: Other Class: Intel Class: TSMC Class: Unknown
Sensitivity            0.23346      0.00000       0.3649      0.83916         0.3474
Specificity            0.98937      0.96972       0.7314      0.59292         0.9399
Pos Pred Value         0.83333      0.00000       0.3816      0.19169         0.7768
Neg Pred Value         0.85008      0.97621       0.7171      0.96974         0.7051
Prevalence             0.18543      0.02309       0.3124      0.10317         0.3759
Detection Rate         0.04329      0.00000       0.1140      0.08658         0.1306
Detection Prevalence   0.05195      0.02958       0.2987      0.45166         0.1681
Balanced Accuracy      0.61142      0.48486       0.5481      0.71604         0.6436
```

In generale, si può dire che il modello di Naive Bayes ha una performance molto bassa, soprattutto considerando che l'accuratezza è inferiore alla No information Rate, ciò significa che il modello non è in grado di fare classificazioni migliori di quelle casuali. Inoltre, il Kappa e molte delle altre misure di performance sono molto basse, questo indica che il modello non è in grado di identificare correttamente molte delle osservazioni.

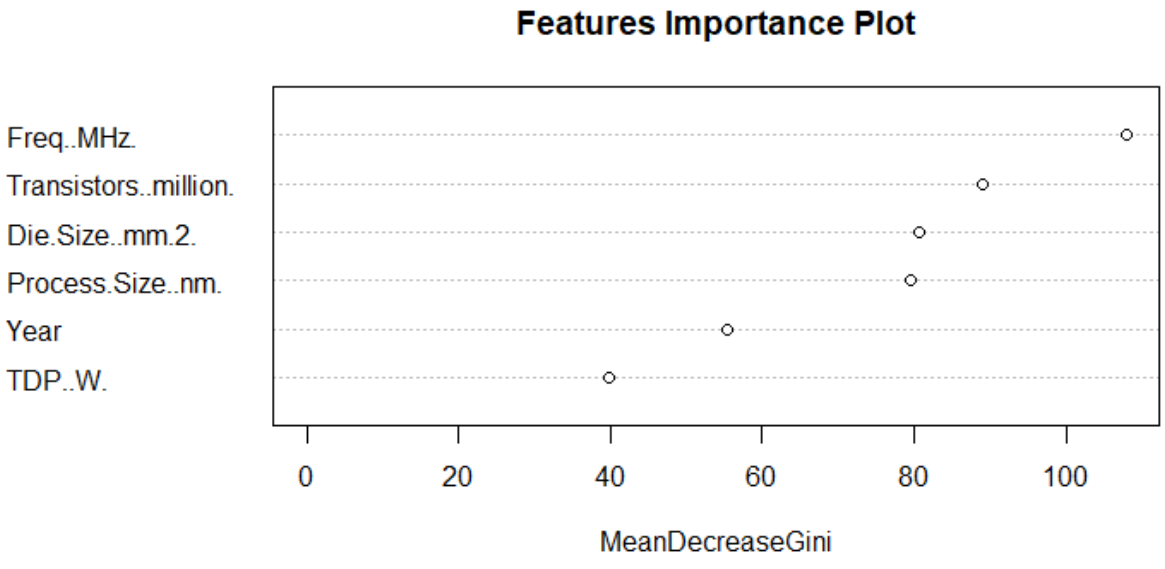
Random Forest:

```
Confusion Matrix and Statistics

          model_rf_under
            GF Other Intel TSMC Unknown
  GF        70     0     1    1       0
  Other      0    37     1    3       0
  Intel      2     2   350   21      39
  TSMC      23    40     5  550       8
  Unknown    0     3    11   11     208

Overall Statistics

                Accuracy : 0.8766
                  95% CI : (0.8581, 0.8935)
    No Information Rate : 0.4228
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8221

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: GF Class: Other Class: Intel Class: TSMC Class: Unknown
Sensitivity            0.73684      0.45122       0.9511      0.9386         0.8157
Specificity            0.99845      0.99693       0.9371      0.9050         0.9779
Pos Pred Value         0.97222      0.90244       0.8454      0.8786         0.8927
Neg Pred Value         0.98097      0.96654       0.9815      0.9526         0.9592
Prevalence             0.06854      0.05916       0.2655      0.4228         0.1840
Detection Rate         0.05051      0.02670       0.2525      0.3968         0.1501
Detection Prevalence   0.05195      0.02958       0.2987      0.4517         0.1681
Balanced Accuracy      0.86765      0.72408       0.9441      0.9218         0.8968
```

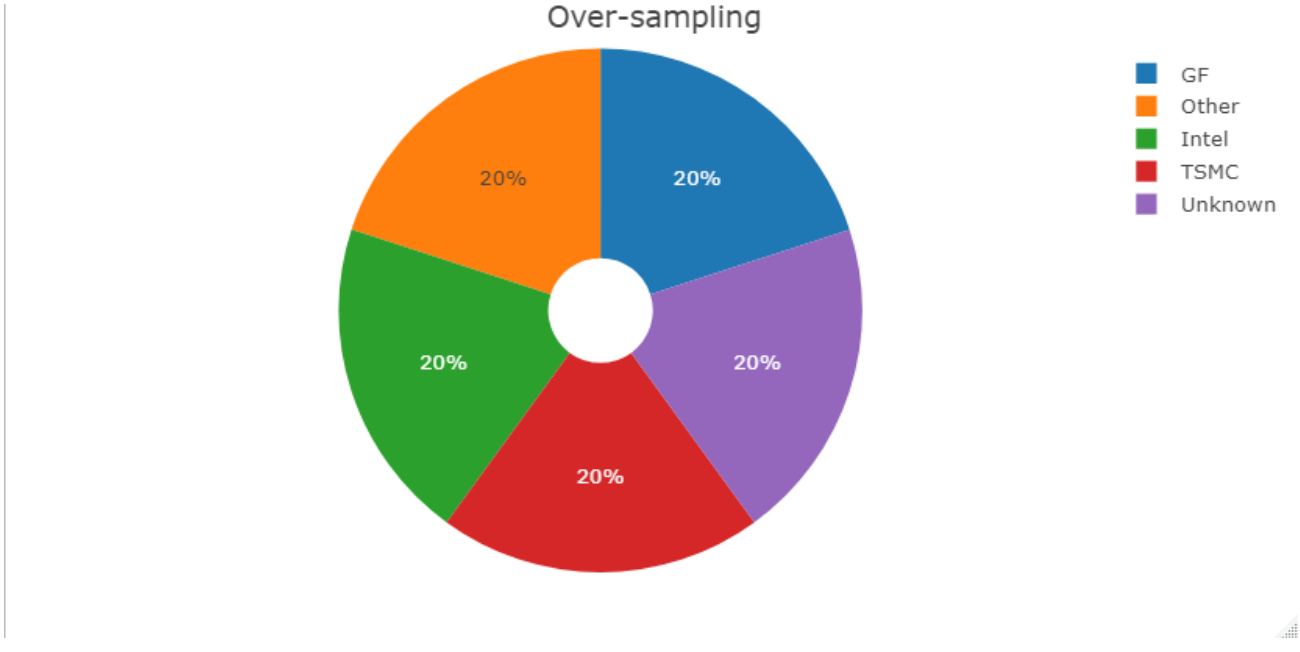
In generale, si può dire che il modello di Random Forest ha una performance eccellente, l'algoritmo di Random Forest è un algoritmo molto potente in grado di gestire la complessità dei dati e di generalizzare bene, questo si traduce in un'accuratezza molto alta e un Kappa molto elevato. Inoltre, tutte le misure di performance mostrate sono molto alte, questo indica che il modello è in grado di identificare correttamente molte delle osservazioni.

Infine, ho creato un grafico per visualizzare l'importanza relativa delle diverse features utilizzate nel modello. Il grafico mostra che la feature meno importante per la classificazione è TDP (Thermal Design Power) ovvero il consumo di energia del processore, mentre la feature più importante è Freq (frequenza del processore). Ciò

significa che la frequenza del processore è stata considerata come una delle feature più influenti per la classificazione delle osservazioni, mentre il consumo di energia del processore non ha avuto un grande impatto sulla capacità del modello di classificare correttamente le osservazioni.



Successivamente ho deciso di utilizzare la tecnica di sovracampionamento casuale, nota anche come "Random Oversampling". Il sovracampionamento casuale consiste nell'aumentare la quantità di dati appartenenti alle classi minoritarie, mediante la ripetizione casuale di alcune delle istanze esistenti. In questo modo si equilibrano le quantità di dati di ogni classe, riducendo così il rischio di sovrastima delle performance del modello. Questa tecnica consente di aumentare la rappresentatività delle classi minoritarie e quindi migliorare la capacità del modello di generalizzare al di fuori del set di dati di addestramento.



Dal grafico è possibile notare che il dataset di training è stato nuovamente bilanciato, poiché ogni classe presenta un numero di istanze pari a quello della classe con il maggior numero di istanze.

A questo punto, ho utilizzato nuovamente i tre algoritmi di classificazione (Alberi di Decisione, Naive Bayes, Random Forest) che avevo già utilizzato in precedenza, per valutare la loro performance e confrontare i risultati tra di loro. In questo modo ho potuto verificare se l'utilizzo della tecnica di sovracampionamento casuale ha avuto un impatto significativo sulle performance dei modelli e se uno degli algoritmi si è dimostrato più efficace degli altri.

Alberi di Decisione:

```
Confusion Matrix and Statistics

          model_dt_over
            GF Other Intel TSMC Unknown
  GF        71     0     0    0       1
  Other      2    36     2    1       0
  Intel      4     2   368    6      34
  TSMC      20    24    12  543      27
  Unknown    0     2    12   10     209

Overall Statistics

                Accuracy : 0.8853
                  95% CI : (0.8673, 0.9016)
     No Information Rate : 0.404
     P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.8349

 Mcnemar's Test P-Value : 3.562e-11

Statistics by Class:

                     Class: GF Class: Other Class: Intel Class: TSMC Class: Unknown
Sensitivity            0.73196       0.56250       0.9340       0.9696         0.7712
Specificity            0.99922       0.99622       0.9536       0.8995         0.9785
Pos Pred Value         0.98611       0.87805       0.8889       0.8674         0.8970
Neg Pred Value         0.98021       0.97918       0.9733       0.9776         0.9462
Prevalence             0.06999       0.04618       0.2843       0.4040         0.1955
Detection Rate         0.05123       0.02597       0.2655       0.3918         0.1508
Detection Prevalence   0.05195       0.02958       0.2987       0.4517         0.1681
Balanced Accuracy      0.86559       0.77936       0.9438       0.9346         0.8748
```

Il modello di Decision Tree mostrato ha un'accuratezza generale del 88,53%, il che significa che il modello classifica correttamente circa il 89% delle osservazioni. La differenza tra l'accuratezza del modello e la No Information Rate è statisticamente significativa, questo indica che il modello è in grado di fare classificazioni migliori di quelle casuali.

In generale si può dire che il modello di Decision Tree ha una performance ottima, l'algoritmo è un algoritmo di facile comprensione e di utilizzo, è in grado di gestire la complessità dei dati e di generalizzare bene, questo si traduce in un'accuratezza molto alta e un Kappa molto elevato. Inoltre, tutte le metriche di performance per ogni classe mostrano valori alti, indicando che il modello è in grado di classificare correttamente le osservazioni per ogni classe.

Naive Bayes:

```
Confusion Matrix and Statistics

          model_nb_over
            GF Other Intel TSMC Unknown
```

```
  GF       59     1     3     9       0
  Other    12     4     2     2      21
  Intel   129     1   104    19     161
  TSMC     64    42   124   141     255
  Unknown   2     3    44     0     184

Overall Statistics

               Accuracy : 0.355
                 95% CI : (0.3298, 0.3808)
    No Information Rate : 0.4481
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1919

 Mcnemar's Test P-Value : <2e-16

Statistics by Class:

                     Class: GF Class: Other Class: Intel Class: TSMC Class: Unknown
Sensitivity            0.22180     0.078431      0.37545      0.8246          0.2963
Specificity            0.98839     0.972285      0.72047      0.6008          0.9359
Pos Pred Value         0.81944     0.097561      0.25121      0.2252          0.7897
Neg Pred Value         0.84247     0.965056      0.82202      0.9605          0.6210
Prevalence             0.19192     0.036797      0.19986      0.1234          0.4481
Detection Rate         0.04257     0.002886      0.07504      0.1017          0.1328
Detection Prevalence   0.05195     0.029582      0.29870      0.4517          0.1681
Balanced Accuracy      0.60510     0.525358      0.54796      0.7127          0.6161
```

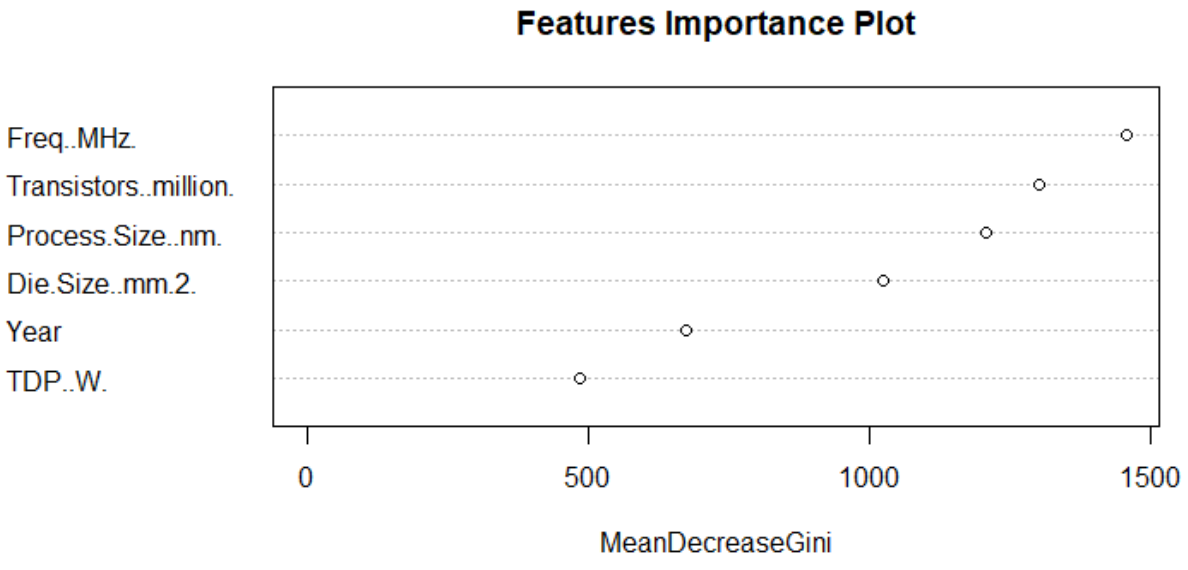
In generale si può dire che il modello di Naive Bayes ha una performance scadente, l'accuratezza generale e il Kappa sono bassi, tutte le metriche di performance per ogni classe mostrano valori bassi, indicando che il modello non è in grado di classificare correttamente le osservazioni per ogni classe.

Random Forest:

```
Confusion Matrix and Statistics

          model_rf_over
            GF Other Intel TSMC Unknown
  GF        72     0     0    0       0
  Other      0    35     2    4       0
  Intel      0     0   408    4       2
  TSMC       0     4     1  615       6
  Unknown    0     1     3    2     227

Overall Statistics

               Accuracy : 0.9791
                 95% CI : (0.9701, 0.9859)
    No Information Rate : 0.4509
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.969

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: GF Class: Other Class: Intel Class: TSMC Class: Unknown
Sensitivity            1.00000      0.87500       0.9855       0.9840          0.9660
Specificity            1.00000      0.99554       0.9938       0.9855          0.9948
Pos Pred Value         1.00000      0.85366       0.9855       0.9824          0.9742
Neg Pred Value         1.00000      0.99628       0.9938       0.9868          0.9931
Prevalence             0.05195      0.02886       0.2987       0.4509          0.1696
Detection Rate         0.05195      0.02525       0.2944       0.4437          0.1638
Detection Prevalence   0.05195      0.02958       0.2987       0.4517          0.1681
Balanced Accuracy      1.00000      0.93527       0.9897       0.9848          0.9804
```

Il modello di Random Forest mostrato in questa classificazione sembra essere molto preciso. La maggior parte delle statistiche presentano valori molto alti, come l'accuratezza (0.9791), per la maggior parte delle classi. Inoltre, l'indice Kappa è anche molto elevato (0.969), il che indica che c'è una forte concordanza tra le predizioni del modello e i valori effettivi. In generale si può dire che questo modello classifichi in modo corretto.

Infine, ho creato un grafico per visualizzare l'importanza relativa delle diverse features utilizzate nel modello. Il grafico mostra che, rispetto alla tecnica di sottocampionamento, l'ordine delle features risulta simile, con la sola eccezione di Die Size e Process Size che si sono invertiti di posizione.



Conclusioni

In conclusione, ho scoperto che utilizzare tecniche di sovracampionamento ha portato a una maggiore accuratezza nella classificazione rispetto alla tecnica di sottocampionamento, soprattutto con gli algoritmi di Decision Tree e Random Forest. Tuttavia, gli algoritmi Naive Bayes e Gaussian Naive Bayes non hanno mostrato un aumento significativo di accuratezza con la tecnica di sovracampionamento.

Ho anche notato che l'algoritmo di Random Forest si è dimostrato essere il più preciso tra tutti gli algoritmi utilizzati. Ciò è stato evidente sia nei risultati ottenuti con la tecnica di sottocampionamento che in quelli ottenuti con la tecnica di sovracampionamento.

In generale, questa analisi ci ha permesso di comprendere l'importanza delle diverse features utilizzate nel modello, come ad esempio la frequenza del processore e la densità di transistori. Inoltre, attraverso l'utilizzo di diverse tecniche di sovracampionamento e algoritmi di classificazione, siamo stati in grado di identificare con precisione il produttore del processore in base alle sue caratteristiche numeriche.