Progetto di Machine Learning

Predizione Qualità Mele

Andrea Falbo
a.falbo7@campus.unimib.it
887525

Damiano Pellegrini
d.pellegrini10@campus.unimib.it
886261

Ruben Tenderini
r.tenderini@campus.unimib.it
879290

Università degli Studi di Milano-Bicocca

Indice

Capitolo 1

Introduzione e Obiettivi

1.1 Obiettivo

L'obiettivo del seguente elaborato è la progettazione di un modello di classificazione in grado di distinguere tra **mele di buona qualità** (etichettate come "good") e **mele di scarsa qualità** (etichettate come "bad"). Il modello utilizzerà le informazioni fornite dal dataset per poter apprendere quali caratteristiche differenziano tale categorie.

1.2 Librerie Utilizzate

Per lo sviluppo del modello e l'analisi dei dati, sono state prese in considerazione le seguenti librerie Python:

- **NumPy**: Libreria utilizzata per la manipolazione efficiente dei dati numerici ed equazioni matematiche veloci.
- **Pandas**: Liberia utilizzata per la gestione e l'analisi dei dati tabulari.
- **Matplotlib** e **Seaborn**: Librerie utilizzate per la visualizzazione dei dati attraverso grafici e heatmap.
- **Scikit-learn**: Libreria utilizzata per:
  - Metriche di valutazione: `accuracy_score`, `classification_report`, e `confusion_matrix`.

  - Preprocessing: `StandardScaler`, `MinMaxScaler`, e `RobustScaler`.
  - Modelli di Machine Learning: `GaussianNB`, `SVC`, e `MLPClassifier`.
  - Strumenti avanzati: `GridSearchCV`, `cross_val_score`, e `train_test_split`.
  - Creazione di pipeline: `make_pipeline`.
  - Ensemble: `VotingClassifier`.

- **Warnings** e **os**: Librerie utilizzate per gestire avvisi e configurazioni del sistema operativo.

- **rand**: Liberia utilizzata per la generazione di numeri pseudo-randomici.

Capitolo 2

Design del Dataset

In questo capitolo viene fornita una descrizione teorica delle scelte effettuate nell' organizzazione del dataset utilizzato nel progetto.

2.1 Descrizione del Dataset

Il dataset utilizzato per questo progetto, denominato **Apple Quality**, contiene 4000 osservazioni. Ogni esempio è descritto dalle seguenti variabili:

- **A_id**: Identificatore univoco per ciascun frutto.
- **Size**: Dimensione del frutto, con valori che variano tra -7.15 e 6.41.
- **Weight**: Peso del frutto, con valori che variano tra -7.15 e 5.79.
- **Sweetness**: Grado di dolcezza del frutto, con valori che variano tra -6.89 e 6.37.
- **Crunchiness**: Texture che indica la croccantezza del frutto, con valori che variano tra -6.06 e 7.62.
- **Juiciness**: Livello di succosità del frutto, con valori che variano tra -5.96 e 7.36.
- **Ripeness**: Stato di maturazione del frutto, con valori che variano tra -5.86 e 7.24.
- **Acidity**: Livello di acidità del frutto, con valori che variano tra -7.01 e 7.4.

- **Quality**: Qualità complessiva del frutto, dove i valori sono "good" per mele di buona qualità, e "bad" per mele di scarsa qualità.

2.2 Ipotesi e Assunzioni

Durante l'analisi e lo sviluppo del modello, sono state effettuate le seguenti ipotesi e osservazioni relative al dataset:

- **Bilanciamento delle classi**: Il dataset presenta due classi (*good* e *bad*) che risultano bilanciate, garantendo una distribuzione equa dei dati tra le categorie.
- **Assenza di valori mancanti**: Non sono presenti valori mancanti nel dataset, rendendo superflui interventi di rimozione o gestione di dati assenti.
- **Nessuna variabile categorica**: Il dataset non contiene variabili categoriche, ad eccezione della variabile target (`quality`), che assume valori binari.

Capitolo 3

Analisi Esplorativa dei Dati

Dopo aver selezionato il dataset e tratto delle assunzioni e ipotesi, sono stati esplorati i dati del al fine di comprendere meglio le sue caratteristiche e prepararlo per il modello di classificazione.

3.1 Visualizzazione Iniziale

Per iniziare, è stata eseguita una visualizzazione dei primi record del dataset con il comando `head()`, che permette di avere un'idea immediata dei dati disponibili. Successivamente, è stata utilizzata la funzione `info()` per ottenere una panoramica delle informazioni generali sul dataset, come il numero di righe, le colonne e i tipi di dati.

A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
0.0	−3.970	−2.512	5.346	−1.012	1.844	0.330	−0.492	good
1.0	−1.195	−2.839	3.664	1.588	0.853	0.868	−0.723	good
2.0	−0.292	−1.351	−1.738	−0.343	2.839	−0.038	2.622	bad
3.0	−0.657	−2.272	1.325	−0.098	3.638	−3.414	0.791	good
4.0	1.364	−1.297	−0.385	−0.553	3.031	−1.304	0.502	good

Tabella 3.1: Visualizzazione degli attributi tramite comando `head()`

3.2 Controllo Valori

Per verificare la varietà dei dati, è stato calcolato il numero di valori distinti per ciascuna colonna del dataset. Questo passaggio ha rivelato che la variabile target `Quality` ha effettivamente due valori distinti ("good" e "bad"), ma inizialmente sono stati riscontrati tre valori diversi. Questo fenomeno è stato dovuto all'ultima riga contenente valori nulli e i crediti dei proprietari del dataset. Dopo aver identificato questa riga con `df.isnull().any(axis = 1)`, è stato deciso di rimuoverla insieme alla colonna `A_id`, in quanto non necessaria per l'analisi. È stata inoltre eseguita una verifica sulla presenza di duplicati, ma non sono stati trovati duplicati nel dataset.

	Count	**Dtype**
A_id	4000	float64
Size	4000	float64
Weight	4000	float64
Sweetness	4000	float64
Crunchiness	4000	float64
Juiciness	4000	float64
Ripeness	4000	float64
Acidity	4001	object
Quality	4000	object

Tabella 3.2: Visualizzazione di informazioni generali tramite comando `info()`

3.3 Casting dei Tipi di Dato

Per ottimizzare l'utilizzo della memoria e mantenere la precisione, i dati numerici sono stati convertiti in tipi di dato più efficienti (`float32`). In particolare, le colonne `Size`, `Weight`, `Sweetness`, `Crunchiness`, `Juiciness`, `Ripeness`, e `Acidity` sono state tutte convertite a `float32`. Inoltre, la variabile target `Quality` è stata trasformata in una variabile numerica binaria: 0 per "bad" e 1 per "good".

	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	Acidity	Quality
0	−3.970 048	−2.512 336	5.346 330	−1.012 009	1.844 900	0.329 840	−0.491 590	1
1	−1.195 217	−2.839 257	3.664 059	1.588 232	0.853 286	0.867 530	−0.722 809	1
2	−0.292 024	−1.351 282	−1.738 429	−0.342 616	2.838 635	−0.038 033	2.621 636	0
3	−0.657 196	−2.271 627	1.324 874	−0.097 875	3.637 970	−3.413 761	0.790 723	1
4	1.364 217	−1.296 612	−0.384 658	−0.553 006	3.030 874	−1.303 849	0.501 984	1

Tabella 3.3: Visualizzazione dati dopo preprocessing tramite comando `head()`

3.4 Bilanciamento delle Classi

È stato osservato che le classi nel dataset sono bilanciate, con una media di 0.5 per la variabile `Quality`. Il bilanciamento della variabile ci garantisce che il modello non sarà influenzato da uno squilibrio tra le classi, eliminando la necessità di applicare tecniche come il re-sampling o l'assegnazione di pesi maggiori alle classi meno rappresentate.

3.5 Matrice di Correlazione

Per esplorare le relazioni tra le variabili numeriche, è stata calcolata la matrice di correlazione Figura 3.1. Della quale, è stato deciso di visualizzare solo una metà per evitare ridondanza. Dalla matrice si può evincere come ci siano quattro correlazioni importanti tra le feature e la "Quality", tre delle quali con correlazione positiva indicandoci come un aumento del valore di queste feature ci porta anche a una probabilità maggiore di avere mele di qualità.

3.6 Distribuzione delle Variabili

Per analizzare la distribuzione delle variabili numeriche, sono stati creati istogrammi con densità (KDE), visualizzabili in Figura 3.2. Questo ci ha permesso di osservare come i dati sono distribuiti e se ci sono eventuali tendenze o anomalie.

3.7 Identificazione degli Outlier

Gli outlier sono stati identificati tramite i boxplot, visualizzabili in Figura 3.3 che mostrano visivamente eventuali valori anomali nelle variabili. Questo passaggio è

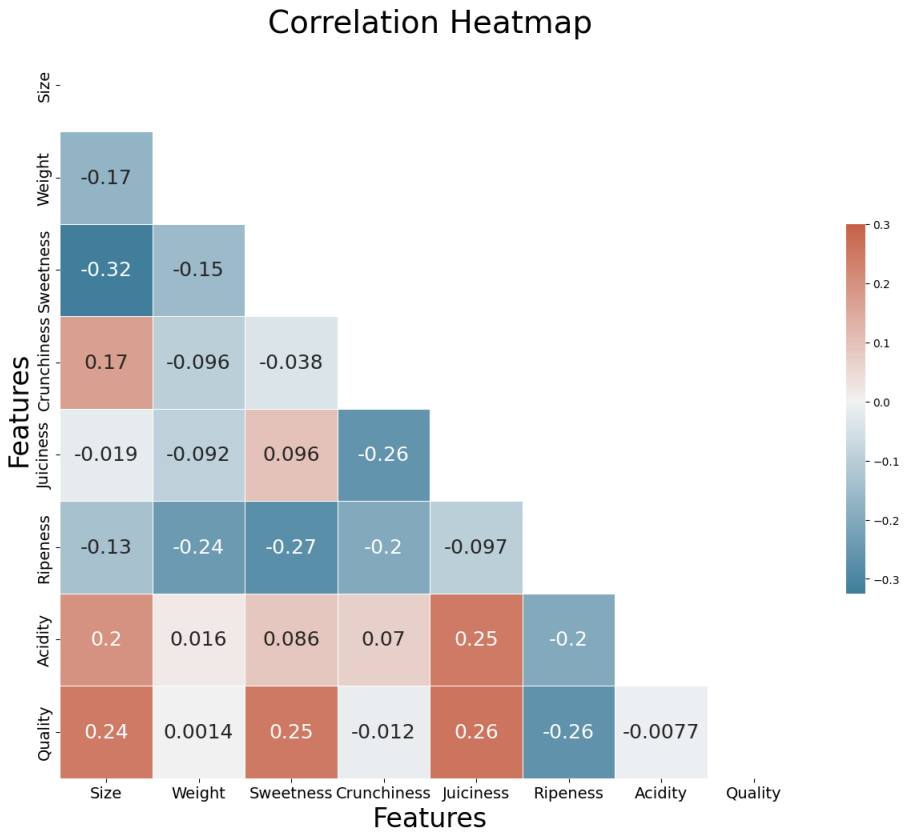

Figura 3.1: Matrice di correlazione delle variabili

fondamentale per determinare se è necessario applicare tecniche di normalizzazione o rimuovere dati fuori scala.

3.8 Normalizzazione dei Dati

Per garantire che tutte le variabili avessero la stessa scala e non influenzassero in modo sproporzionato il modello, sono stati applicati diversi metodi di normalizzazione:

- **RobustScaler**: Usato per rimuovere gli outlier presenti nel dataset.
- **StandardScaler**: Applicato per ottenere una distribuzione normale (gaussiana) delle variabili.

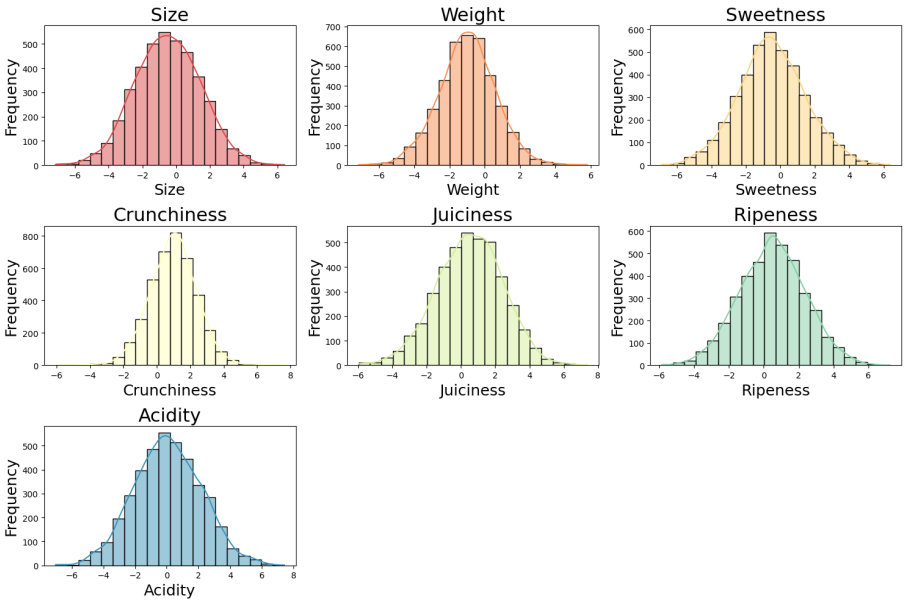

Figura 3.2: Distribuzione delle variabili nel dataset

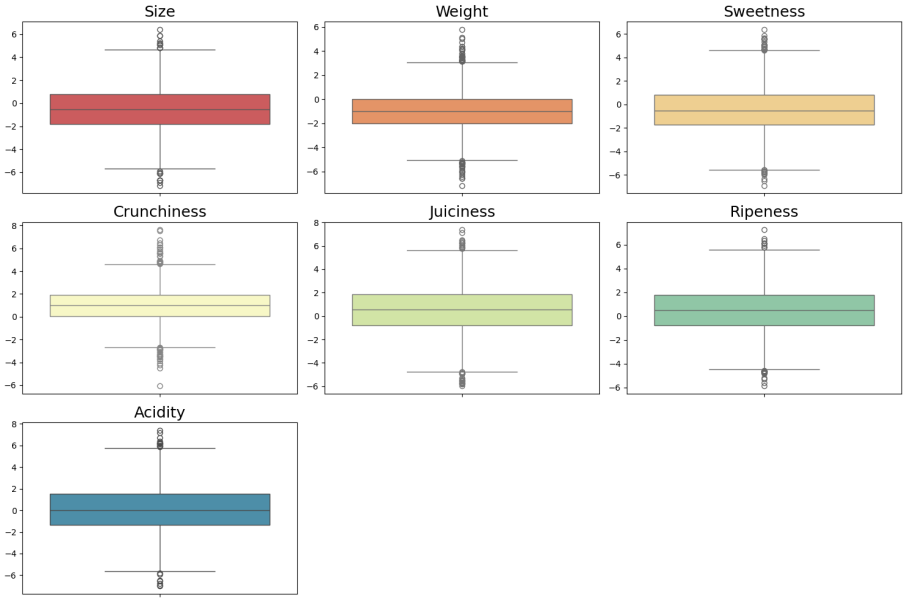

Figura 3.3: Boxplot per l'identificazione degli outlier

Anche se la normalizzazione con `MinMaxScaler` sarebbe stata un'opzione, è stato scelto di non utilizzarla in quanto per i modelli che successivamente verranno utilizzati è sufficiente una distribuzione gaussiana.

In Tabella 3.4 è possibile osservare i dati prima di essere stati normalizzati e aver rimosso i valori anomali, mentre nella Tabella 3.5 è possibile osservarli dopo tali procedimenti.

	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
Size	4000.0	−0.503 015	1.928 058	−7.151 703	−1.816 765	−0.513 703	0.805 526	6.406 367
Weight	4000.0	−0.989 547	1.602 507	−7.149 848	−2.011 770	−0.984 737	0.030 976	5.790 714
Sweetness	4000.0	−0.470 479	1.943 441	−6.894 485	−1.738 425	−0.504 758	0.801 922	6.374 916
Crunchiness	4000.0	0.985 478	1.402 757	−6.055 058	0.062 764	0.998 249	1.894 234	7.619 852
Juiciness	4000.0	0.512 118	1.930 287	−5.961 897	−0.801 286	0.534 219	1.835 976	7.364 403
Ripeness	4000.0	0.498 277	1.874 426	−5.864 599	−0.771 677	0.503 445	1.766 212	7.237 837
Acidity	4000.0	0.076 877	2.110 271	−7.010 539	−1.377 424	0.022 609	1.510 493	7.404 736
Quality	4000.0	0.501 000	0.500 062	0.000 000	0.000 000	1.000 000	1.000 000	1.000 000

Tabella 3.4: Visualizzazione **prima** della rimozione outliers e normalizzazione dei dati

	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
Size	4000.0	1.335 144	1.000 125	−3.448 816	−0.681 470	−0.005 544	0.678 768	3.584 043
Weight	4000.0	1.907 349	1.000 125	−3.844 645	−0.637 970	0.003 002	0.636 909	4.231 561
Sweetness	4000.0	−8.821 488	1.000 124	−3.305 895	−0.652 505	−0.017 641	0.654 797	3.522 747
Crunchiness	4000.0	8.583 068	1.000 125	−5.019 697	−0.657 868	0.009 106	0.647 917	4.730 115
Juiciness	4000.0	3.814 697	1.000 125	−3.354 335	−0.680 504	0.011 451	0.685 921	3.550 325
Ripeness	4000.0	1.430 511	1.000 126	−3.394 996	−0.677 601	0.002 757	0.676 523	3.595 980
Acidity	4000.0	−9.536 744	1.000 125	−3.358 955	−0.689 240	−0.025 720	0.679 437	3.472 910
Quality	4000.0	5.010 000	0.500 062	0.000 000	0.000 000	1.000 000	1.000 000	1.000 000

Tabella 3.5: Visualizzazione **dopo** rimozione outliers e normalizzazione dei dati

Capitolo 4

Modelli di Machine Learning

In questo capitolo verranno presentati i modelli di machine learning sviluppati per l'analisi della qualità delle mele. I modelli sono stati scelti cercando di includere approcci con diverse caratteristiche e complessità, al fine di confrontarne le prestazioni. Sono stati selezionati tre modelli: **Support Vector Machine**, **Naive Bayes** e **Multi Layer Perceptron**.

4.1 Support Vector Machine

Il modello di **Support Vector Machine** (SVM) si basa sulla ricerca di un iperpiano che separi al meglio le classi in uno spazio ad alta dimensione, cercando di massimizzare il margine tra le due classi.

Tale modello è stato scelto per la sua capacità di gestire problemi di classificazione complessi attraverso la ricerca di un iperpiano ottimale che separi le classi. L'implementazione utilizzata è quella fornita dalla libreria **scikit-learn**.

Il modello è stato configurato con un **kernel RBF** (Radial Basis Function), un parametro di regolarizzazione $C = 100$, e un parametro $\gamma = 0.1$, specificati nel seguente codice:

Listing 4.1: Configurazione del Support Vector Machine

```
"svm": SVC(
    kernel="rbf",           # Radial basis function kernel
    C=100,                  # Regularization parameter
    gamma=0.1,              # Kernel coefficient
    probability=True,       # Enable probability estimates
    random_state=RANDOM_STATE, # Ensure reproducibility
)
```

Il **kernel RBF** è particolarmente adatto per dati non linearmente separabili, poichè consente di trasformare lo spazio delle feature in uno spazio a dimensionalità superiore, dove le classi possono essere separate linearmente.

4.2 Naive Bayes

Questo modello di machine learning si basa sul **teorema di Bayes** e sull'assunzione "naive" ("banale" o "ingenua") di indipendenza condizionale tra ogni coppia di features, dato il valore della variabile di classe.
In particolare è stata utilizzata l'implementazione (fornita da scikit-learn) denominata **GaussianNB**, dove si assume che la probabilità di una feature sia gaussiana:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Dove i parametri σ_y e μ_y, che rappresentano rispettivamente la media e la deviazione standard per ciascuna feature x_i, condizionata sulla classe y, sono stimati in base al metodo della massima verosimiglianza.

Listing 4.2: Configurazione del Naive Bayes

```
"nb": GaussianNB()
```

Questo modello è stato utilizzato per la sua semplicità e leggerezza che permettono una rapida esecuzione. Inoltre fornisce una buona base di confronto per modelli più complessi.

4.3 Multi Layer Perceptron

Il **Multi Layer Perceptron** (MLP) è un modello di rete neurale artificiale a strati, ideale per problemi di classificazione non lineari. È stato scelto per la sua flessibilità nel modellare relazioni complesse tra le feature e le classi.

Tale modello è stato implementato utilizzando la libreria **scikit-learn** con una configurazione che prevede tre hidden layer, ciascuno contenente 10 neuroni. Il modello è stato configurato per un massimo di 750 iterazioni e uno stato casuale specificato per garantire la riproducibilità:

Listing 4.3: Configurazione del Multi-layer Perceptron

```
"mlp": MLPClassifier(
    hidden_layer_sizes=(10, 10, 10),
    max_iter=750,
    random_state=RANDOM_STATE,
)
```

La funzione di attivazione utilizzata è la **ReLU** (Rectified Linear Unit):

$$\text{ReLU}(x) = \max(0, x)$$

mentre l'ottimizzazione è effettuata utilizzando l'algoritmo **Adam** (Adaptive Moment Estimation), che combina i vantaggi della discesa del gradiente con momento e del rate di apprendimento adattivo. La funzione di costo minimizzata è la **cross-entropy loss**:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

dove $y_{i,c}$ è il valore vero e $\hat{y}_{i,c}$ è la probabilità predetta per la classe c dell'osservazione i-esima.

Capitolo 5

Esperimenti e Validazione

In questo capitolo descriveremo i metodi adottati per la costruzione e la validazione dei modelli, fornendo dettagli sul processo di separazione del dataset, sull'inizializzazione dei modelli e sulle tecniche utilizzate per migliorarne l'efficienza e l'affidabilità.

5.1 Separazione del Dataset

Per prima cosa è stata eseguita una **separazione del dataset**: 80% dei dati sono stati assegnati alla fase di **training**, mentre il restante 20% è stato utilizzato per la fase di test. Inoltre è stato utilizzato un seed randomico per la separazione dei dati e per la successiva implementazione dei vari modelli.

Listing 5.1: Dataset splitting

```
RANDOM_STATE = random.randint(0, 1337)  # Random seed

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE
)
```

5.2 Grid Search

Successivamente, sono stati inizializzati i modelli mostrati nel capitolo precedente all'interno di un dizionario, per facilitare la fase di esecuzione e ottimizzazione.

Tramite l'utilizzo di una **grid search** è poi avvenuta un'ottimizzazione degli iperparametri. Questo algoritmo esegue una ricerca esaustiva degli migliori valori degli iperparametri all'interno di un insieme prefefinito.

Listing 5.2: Grid search

```
grid_results = {}
for model_key, model in classifiers.items():
    grid_search = GridSearchCV(
        model,
        param_grid[model_key],
        cv=5,
        n_jobs=-1
    )
    grid_search.fit(X_train, y_train)
    grid_results[model_key] = grid_search.best_params_
    classifier[model_key] = grid_search.best_estimator_
```

Di seguito sono mostrati gli insiemi di valori sui quali è stata eseguita l'ottimizzazione.

Listing 5.3: Parametri di ricerca della grid search

```
param_grid = {
    "svm": {
        "C": [0.1, 1, 10, 100],
        "kernel": ["linear", "rbf", "poly"],
        "gamma": ["scale", "auto", 0.1, 1],
    },
    "nb": {},
    "mlp": {
        "hidden_layer_sizes": [
            (8, 8, 8),
            (10, 10, 10),
            (12,12,12)
        ],
        "max_iter": [500, 750, 1000]
    },
}
```

5.3 Cross Validation

Le configurazioni dei modelli ottenute tramite la grid search sono state successivamente valutate utilizzando la tecnica della **cross-validation**, un metodo robusto per stimare le prestazioni del modello. In particolare, i dati di training sono stati suddivisi in **5 fold**, cioè in 5 sottoinsiemi di dimensioni uguali. In ciascuna iterazione, uno dei fold veniva utilizzato come test set, mentre gli altri 4 venivano impiegati per addestrare il modello. Questo processo è stato ripetuto per ogni fold, garantendo che ciascun campione del dataset fosse utilizzato sia per l'addestramento che per il testing. Grazie a questo approccio, è stato possibile ottenere una **stima più affidabile** e meno influenzata dalla variabilità dei dati.

Listing 5.4: Parametri di ricerca della grid search

```
cv_results = {}
for model_name, model in classifiers.items():
    cv_score = cross_val_score(
        model,
        X_train,
        y_train,
        cv=5)
    cv_results[model_name] = cv_score.mean()
cv_results = pd.DataFrame(cv_results.items())
```

Capitolo 6

Analisi dei Risultati

In questo capitolo andremo ad analizzare i risultati e fornire punti di osservazione importanti atti a capire la capacità dei vari modelli di effettuare una predizione affidabile e precisa.

6.1 Analisi Training

Nella Tabella 6.1 vengono riportate le metriche ottenute dai modelli durante la fase di allenamento tramite **cross validation**. Le metriche includono la precisione media su tutte le fold, evidenziando la capacità dei modelli di generalizzare sui dati di test.

Model	Precision
Support Vector Machine	0.907500
Naive-Bayes	0.751563
MultiLayer Perceptron	0.929063

Tabella 6.1: Metriche di allenamento tramite `cross validation` dei modelli.

Possiamo osservare come **SVM** e **MLP** hanno ottenuto ottime prestazioni in termini di precisione media durante la fase di allenamento. In particolare, il modello MLP ha raggiunto la precisione più alta con 0.929063, seguito da SVM con 0.907500. Al contrario, il modello **NB** ha mostrato una precisione significativamente inferiore (0.751563), suggerendo che le sue capacità di generalizzazione sui dati di test sono limitate rispetto agli altri due modelli.

6.2 Report di classificazione

Dopo aver valutato la fase di training dei 3 modelli, abbiamo eseguito delle valutazioni sulla fase di test per i 3 modelli. Per farlo, abbiamo osservato metriche fondamentali come **precision**, **recall** e **F1-score**.

6.2.1 Precision

La **precision** misura quante istanze sono state correttamente identificate come positive rispetto a tutte le istanze predette come positive. In altre parole, indica la capacità del modello di evitare falsi positivi. La sua formula è la seguente:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

6.2.2 Recall

La **recall**, nota anche come sensibilità o *true positive rate*, misura la capacità del modello di identificare correttamente tutte le istanze positive. Si calcola come:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

6.2.3 F1-Score

L'**F1-score** è la media armonica tra *precision* e *recall*, offrendo un equilibrio tra le due metriche. Nel nostro contesto, dato che le due classi sono bilanciate. non è particolarmente rilevante. Si calcola come:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

6.2.4 Support Vector Machine

La Tabella 6.2 mostra le metriche di classificazione per il modello Support Vector Machine. Il modello ha raggiunto un'accuratezza complessiva del 91%, con performance bilanciate tra le due classi.

Support Vector Machine				
Class	**Precision**	**Recall**	**F1-Score**	**Support**
Bad Apples	0.90	0.93	0.91	404
Good Apples	0.93	0.89	0.91	396
Accuracy			0.91	800
Macro Avg	0.91	0.91	0.91	800
Weighted Avg	0.91	0.91	0.91	800

Tabella 6.2: Report di classificazione del modello Support Vector Machine.

6.2.5 Naive-Bayes

La Tabella 6.3 evidenzia le metriche per il modello Naive-Bayes, che mostra un'accuratezza del 75%, inferiore rispetto agli altri modelli, con prestazioni simili su entrambe le classi.

Naive-Bayes				
Class	**Precision**	**Recall**	**F1-Score**	**Support**
Bad Apples	0.75	0.76	0.76	404
Good Apples	0.75	0.74	0.75	396
Accuracy			0.75	800
Macro Avg	0.75	0.75	0.75	800
Weighted Avg	0.75	0.74	0.75	800

Tabella 6.3: Report di classificazione del modello Naive-Bayes.

6.2.6 Multilayer Perceptron

La Tabella 6.4 presenta le metriche di classificazione per il modello Multilayer Perceptron, che ha raggiunto un'accuratezza del 93%, risultando il più performante tra quelli analizzati.

6.3 Confronto dei modelli

Di seguito vengono riportati i grafici comparativi per analizzare visivamente le performance dei modelli. Per ottenere una migliore comprensione delle differenze tra i modelli, sono stati realizzati diversi plot, tra cui grafici a barre per confrontare

Multilayer Perceptron				
Class	**Precision**	**Recall**	**F1-Score**	**Support**
Bad Apples	0.93	0.94	0.93	404
Good Apples	0.94	0.93	0.93	396
Accuracy			0.90	800
Macro Avg	0.93	0.93	0.93	800
Weighted Avg	0.93	0.93	0.93	800

Tabella 6.4: Report di classificazione del modello Multilayer Perceptron.

le metriche principali (**precision**, **recall**, **F1-score**), matrici di confusione per valutare la distribuzione degli errori di classificazione e curve **ROC** per visualizzare le performance dei modelli in termini di compromesso tra *true positive rate* e *false positive rate*.

6.3.1 Bar Plot

La Figura 6.1 presenta un confronto sintetico delle metriche principali (precision, recall e F1-score) tra i modelli utilizzando un bar plot. Questo grafico facilita l'interpretazione delle differenze nelle performance.

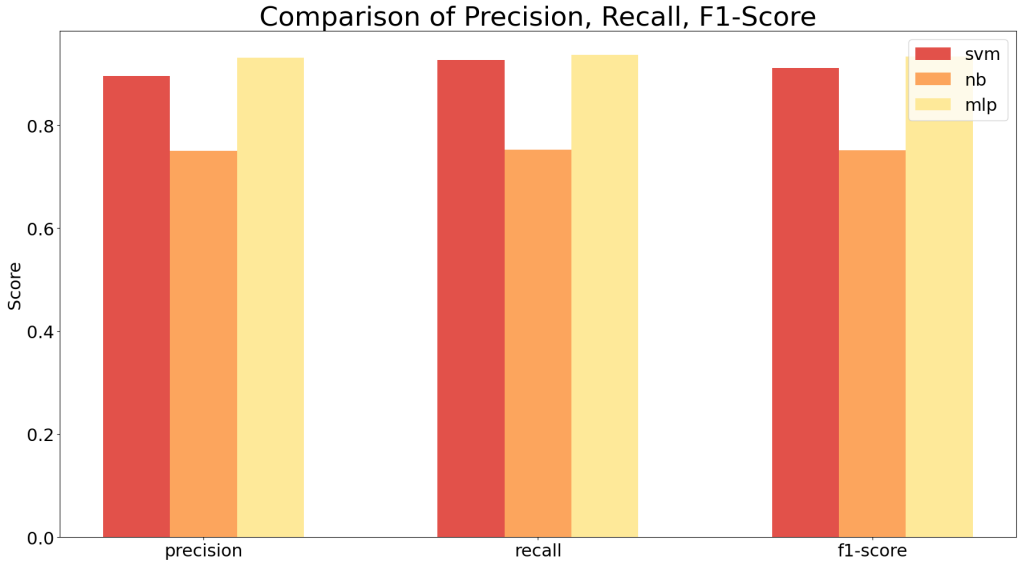

Figura 6.1: Confronto dei modelli tramite bar plot delle metriche principali.

6.3.2 Matrice di Confusione

La Figura 6.2 illustra le matrici di confusione per i tre modelli, permettendo di osservare visivamente gli errori di classificazione commessi.

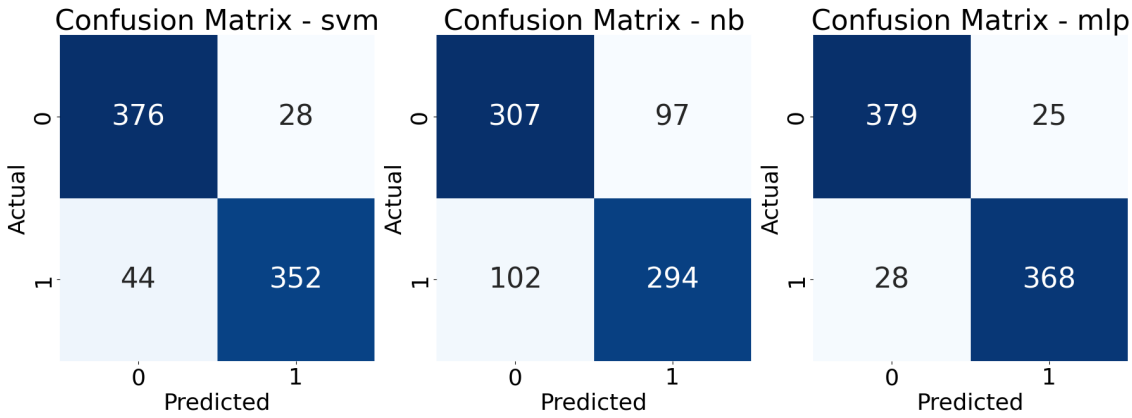

Figura 6.2: Confronto dei modelli tramite matrice di confusione.

6.3.3 Curva ROC

Infine, la Figura 6.3 mostra il confronto tra i modelli in termini di curva ROC, evidenziando la capacità di discriminazione tra le classi. Il Multilayer Perceptron risulta avere l'area sotto la curva (AUC) più elevata.

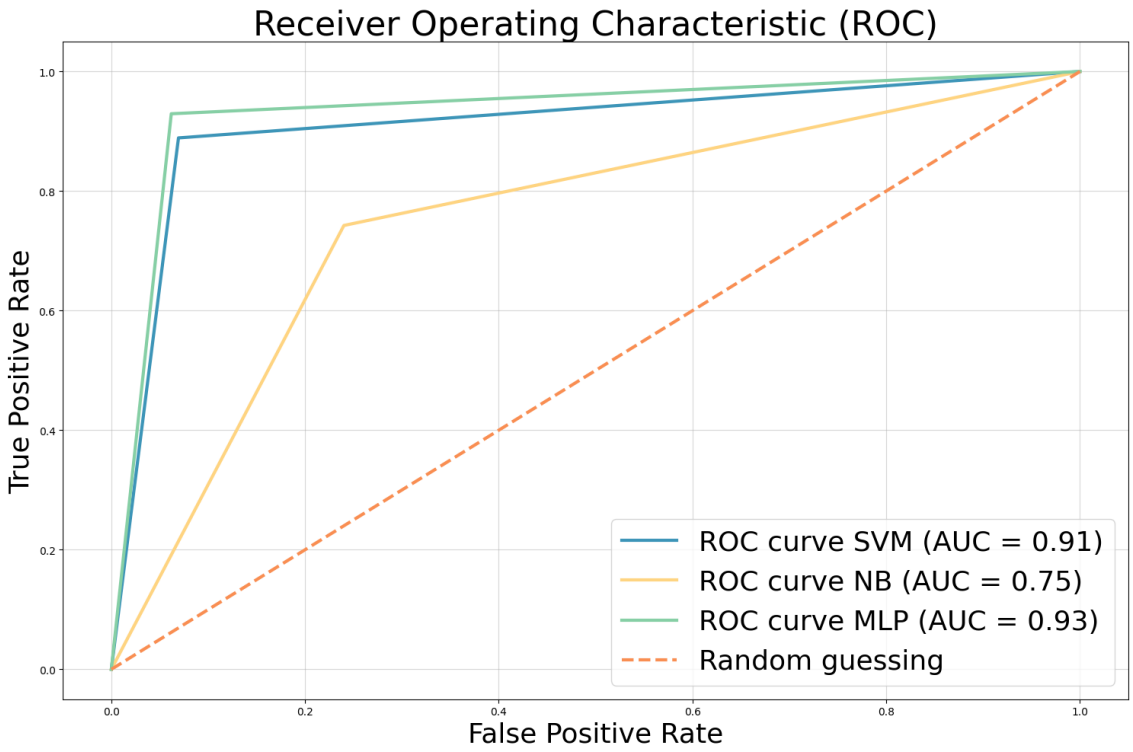

Figura 6.3: Confronto dei modelli tramite curva ROC

Capitolo 7

Conclusioni

Dai risultati ottenuti durante l'analisi, è emerso che il **Multi Layer Perceptron (MLP)** è stato il classificatore con le migliori prestazioni complessive. Il successo del MLP è attribuibile alla sua capacità di catturare pattern non lineari nei dati grazie alla presenza di strati nascosti. Questi risultati lo collocano leggermente sopra il **Support Vector Machine** (**SVM**), il quale ha ottenuto punteggi simili. È stato però osservato che, nelle varie run eseguite, l'SVM è sempre risultato più consistente nei punteggi rispetto al MLP, che ha oscillato tra 0.9 e 0.93, mentre l'SVM ha mantenuto prestazioni più stabili e prossime al valore massimo. Il **Naive Bayes** si è rivelato significativamente inferiore in termini di prestazioni.