

Sommario del documento

1	Introduzione al problema.....	2
2	Il mio progetto.....	3
3	Fonte e Formato dei Dati.....	4
4	Data Cleaning Utilizzati	5
5	Data Training.....	6
6	Risultati, Confronto tra modelli e Analisi	8
7	Dataset nel Dettaglio.....	12
8	Modifiche al Dataset	15
9	Data Training, Risultati e Confronti finali.....	19
10	Interattività	22

1 Introduzione al problema

La potabilità dell'acqua è un problema non tanto considerato nel mondo odierno in quanto l'accesso nei paesi più sviluppati è pressoché garantito. Infatti, secondo un rapporto fatto dall'UNESCO nel 2017, 5,3 miliardi di persone utilizzava servizi di acqua potabile gestiti in modo sicuro, è ben il 71% della popolazione mondiale all'epoca, però rimane un pesante 29% composto dai restanti 2,2 miliardi di persone, inoltre, 4,2 miliardi di persone (il 55%) non disponeva di servizi igienico-sanitari gestiti in modo sicuro.

La situazione si fa critica se andiamo a considerare i dati dei paesi più poveri, dato che solo il 10% della popolazione mondiale assorbe il 52% del reddito globale, e il 10% più ricco emette il 50% della CO₂, contribuendo al cambiamento climatico, di conseguenza andando a danneggiare le sorgenti naturali d'acqua, lasciando la restante fetta di popolazione mondiale a subirne le conseguenze più ardue, per esempio, in Africa subsahariana, il 63% della popolazione non ha accesso a servizi di acqua potabile gestiti in modo sicuro, e il 75% non ha accesso a servizi igienico-sanitari gestiti in modo sicuro. E in questo caso le donne hanno anche la peggio perché sono quelle incaricate di farsi i “viaggi” di andata-ritorno dalla sorgente fino a casa.

L'agenda 2030 dell'UNESCO dà occhio a questo problema, [l'obiettivo numero 6](#) mira a garantire a tutti l'accesso a servizi di acqua e di igiene adeguati e a migliorare la gestione sostenibile delle risorse idriche. Per raggiungere questo obiettivo è necessario un impegno congiunto delle varie nazioni più sviluppate, di organizzazioni internazionali, della società civile in sé, del settore privato e delle comunità locali; al fine di promuovere politiche, investimenti, innovazioni e pratiche che valorizzino l'acqua e ne assicurino un uso efficiente, ed equo

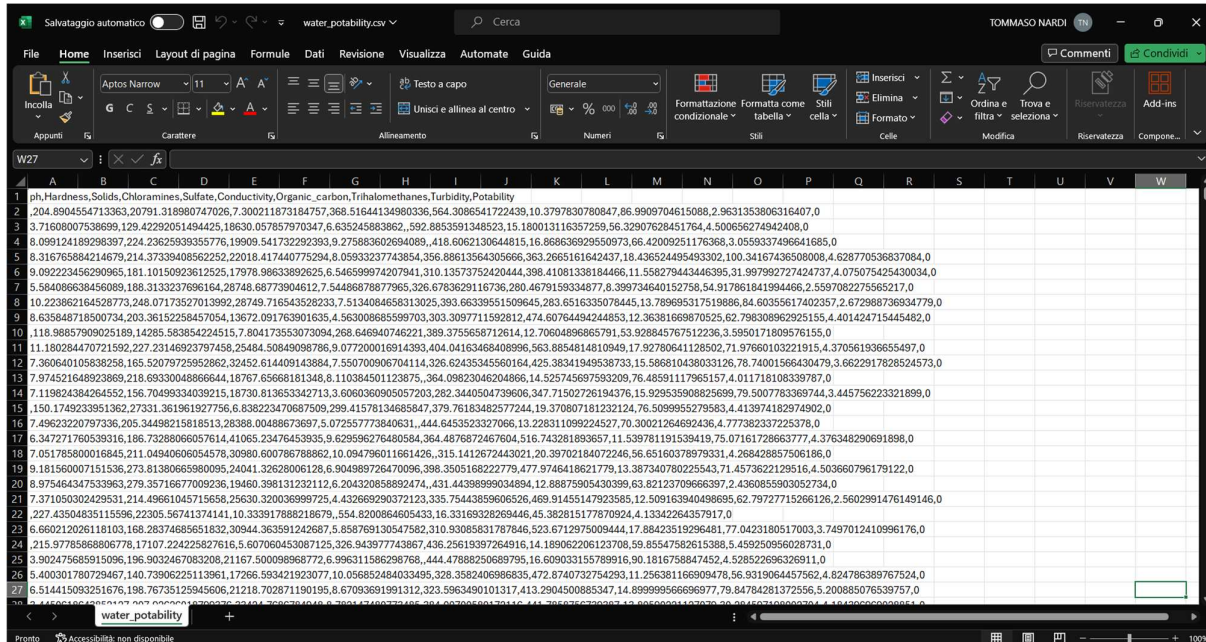
2 Il mio progetto

Ho scelto di trattare questa tematica nel mio piccolo progetto per il corso di ML in quanto è un problema che può portare a serie conseguenze se si continua per questa strada.

Il fine ultimo del progetto è quello di creare un buon modello di IA capace di dirci se l'acqua che stiamo esaminando attualmente è potabile o meno, in modo tale che, finché si possiedono i dati, tutti possano vedere se l'acqua che hanno davanti è potabile o meno, questo progetto è un po' "ambizioso" ma è valsa la pena provarci.

3 Fonte e Formato dei Dati

Il dataset che andremo ad utilizzare è stato trovato su [Kaggle](#), ma ha origine da una repository pubblica su [Github](#). C'è **molto** da dire su questo dataset, ma ogni cosa a suo tempo, andiamo prima a vedere il contenuto del file “water_potability.csv”:



Come possiamo ben vedere, la premessa è buona, ho scelto questo dataset in particolare perché è uno dei pochi che combinano il tema della potabilità dell'acqua con il task di regressione. Elenchiamo le colonne con tanto di descrizione:

ph: Una grandezza fisica che indica se una sostanza è acida (<7), neutra(=7) o basica (>7)

Durezza (Hardness): Calcio e Magnesio nell'acqua, misurato in mg/L (milligrammi/litro)

Solidi (Solids): Solidi Disciolti nell'acqua misurati in ppm (parti per milione)

Cloramine (Chloramines): Cloramina (un agente disinfettante) nell'acqua misurata in ppm

Solfato (Sulfate): Solfato nell'acqua in mg/L

Conducibilità (Conductivity): Capacità di condurre elettricità nell'acqua misurata in microsiemens per centimetro ($\mu\text{S}/\text{cm}$)

Carbonio organico totale (Organic_Carbon): Il carbonio nell'acqua misurato in ppm

Trialometani (Trihalomethanes): Trialometani (sottoprodotti della disinfezione) nell'acqua misurati in microgrammi per litro ($\mu\text{g}/\text{L}$)

Torbidità (Turbidity): Torbidità (la capacità dell'acqua di diffondere la luce) misurata in Unità Nefelometriche di Torbidità (NTU)

4 Data Cleaning Utilizzati

La prima cosa che è venuta da fare è stata quella di esaminare la consistenza delle entry nel dataset dopo il caricamento, cosa facilmente eseguibile con questi comandi (script-modelli):

```
# Carica il dataset
dataset = pd.read_csv("water_potability.csv")
# Stampa dei valori nel dataset
dataset.info()
```

Che mi ha dato questo in output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   ph                     2785 non-null   float64
1   Hardness               3276 non-null   float64
2   Solids                 3276 non-null   float64
3   Chloramines            3276 non-null   float64
4   Sulfate                2495 non-null   float64
5   Conductivity           3276 non-null   float64
6   Organic_carbon         3276 non-null   float64
7   Trihalomethanes       3114 non-null   float64
8   Turbidity              3276 non-null   float64
9   Potability             3276 non-null   int64   
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

Viene subito all'occhio che abbiamo un bel problema: mancano dei dati, più precisamente mancano dei dati per “ph”; “Sulfate” e “Trihalomethanes”. Ora abbiamo davanti a noi un bivio, o cancelliamo queste entry con dati mancanti, andando però a rischiare che ne cancelliamo una buona fetta perché potrebbero essere dati mancanti a caso, non per forza tutti insieme...oppure effettuiamo l'imputazione.

Nel codice ho preposto, in ogni script che deve caricare il dataset, una scelta per l'esecuzione:

```
# Di questi sceglierne solo uno! Il primo cancella le entry con dati mancanti, la seconda invece fa l'imputazione con media
dataset = dataset.dropna()
# dataset.fillna(dataset.mean(), inplace=True)
```

.dropna() cancellerà le entry con i dati mancanti, mentre .fillna(mean) invece effettuerà l'imputazione basata su media. Per tutto il resto di questo documento utilizzerò dati che provengono dal dataset imputato tramite media e mi riferirò ad esso come “dataset imputato” da qui in avanti.

5 Data Training

Il dataset è stato diviso tra dati di training e dati di test:

```
# Divisione del dataset in set di addestramento e test, 20% Test, 80% Training
X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.2, random_state=42)
```

E poi è stato usato per addestrare 4 modelli per decretare il migliore tra loro:

- Decision Tree: Un modello che rappresenta una sequenza di decisioni in forma di struttura ad albero. Ogni nodo interno dell'albero rappresenta una decisione basata su una feature e ogni foglia rappresenta il risultato di questa catena di nodi e decisioni
- Random Forest: Usa tanti Decision Tree insieme, addestrati ognuno su un sottoinsieme diverso, la predizione del Random Forest deriva dal voto di maggioranza tra i vari Decision Tree al suo interno
- Logistic Regression: Modella la probabilità che un'entry appartenga a una 0/1 specifica. Utilizza la funzione logistica per mappare la somma pesata delle caratteristiche di input in un valore compreso tra 0 e 1
- Naive Bayes: Calcola la probabilità di ciascuna feature predittiva (detta anche attributo o segnale) dei dati che appartengono a ciascuna classe, per fare una predizione della distribuzione di probabilità su tutte le classi

Ora andiamo a vedere il codice:

```
# Lista dei modelli che useremo, nell'array "models"
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42),
    "Naive Bayes": GaussianNB()
}

# Per ogni modello che abbiamo definito...
for name, model in models.items():

    # Addestralo
    model.fit(X_train, y_train)

    # Effettua la predizione, y_pred è un array binario di tutte le previsioni, IN ORDINE
    y_pred = model.predict(X_test)
```

E ora andiamo il codice per il calcolo e la stampa delle metriche, che saranno la nostra giuria per vedere quale modello ha le migliori prestazioni su questo dataset e, soprattutto, con quanta efficacia:

```
# Calcolo delle varie metriche confrontando il valore reale con quello predetto (0 o 1)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

#Stampa di tutte le metriche studiate nel corso, con fino a 9 cifre dopo la virgola
print(f"\nMetriche di{name}:")
print(f"Accuracy: {accuracy:.9f}")
print(f"Precision: {precision:.9f}")
print(f"Recall: {recall:.9f}")
print(f"F1-Score: {f1:.9f}")
print("Confusion Matrix:")
print(cm)
```

6 Risultati, Confronto tra modelli e Analisi

I risultati che ora mostro sono stati copiati dalla console di esecuzione dello script:

Metriche di Decision Tree:

- Accuracy: 0.577743902
- Precision: 0.441281139
- Recall: 0.508196721
- F1-Score: 0.472380952
- Confusion Matrix:

```
[[255 157]
```

```
[120 124]]
```

Metriche di Random Forest:

- Accuracy: 0.678353659
- Precision: 0.609271523
- Recall: 0.377049180
- F1-Score: 0.465822785
- Confusion Matrix:

```
[[353 59]
```

```
[152 92]]
```

Metriche di Logistic Regression:

- Accuracy: 0.628048780
- Precision: 0.000000000
- Recall: 0.000000000
- F1-Score: 0.000000000
- Confusion Matrix:

```
[[412 0]
```

```
[244 0]]
```

Metriche di Naive Bayes:

- Accuracy: 0.631097561
- Precision: 0.509615385
- Recall: 0.217213115
- F1-Score: 0.304597701
- Confusion Matrix:

```
[[361 51]
```

```
[191 53]]
```


Dunque, a prima vista si direbbe che il modello migliore sia Random Forest, basandoci sull'Accuracy, ci basiamo su di essa perché noi vogliamo un modello preciso, il tasso di errore deve essere minimo.

Però una cosa ho considerato deludente, e la mostrerò usando dei plot della Learning Curve e della curva ROC creati con questo codice:

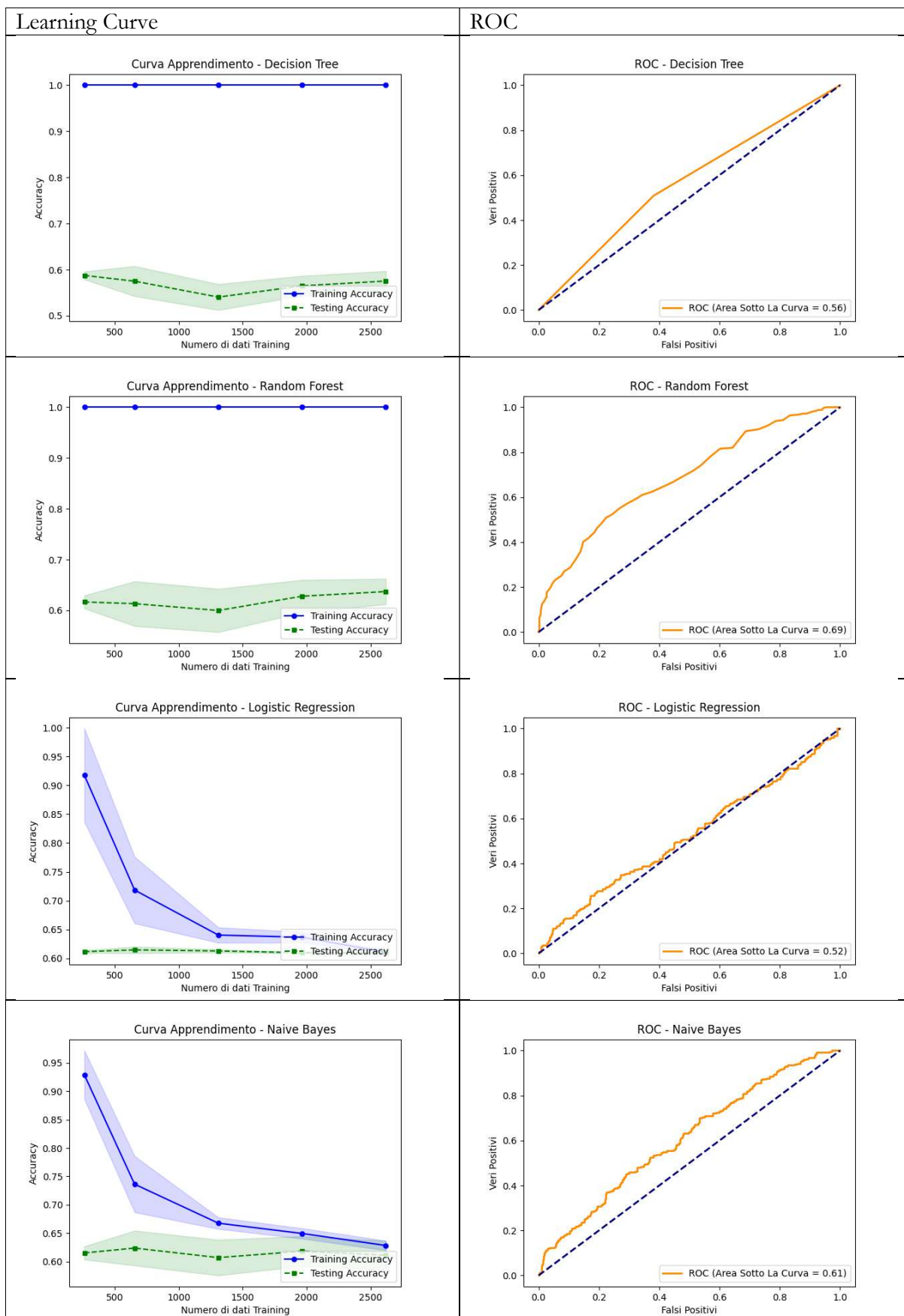
```
# Creazione Immagine della Curva di Apprendimento (come si comporta il modello al variare della dimensione
# del dataset nel corso del tempo usando la cross-validation delle 5 train_sizes
# ricordare che questa curva fa riferimento all'accuracy, come specificato in "scoring")
train_sizes, train_scores, test_scores = learning_curve(
    model, X, y, cv=5, scoring='accuracy', train_sizes=[0.1, 0.25, 0.5, 0.75, 1.0]
)

train_mean = train_scores.mean(axis=1) #Media e Deviazione su dati training, più alta la deviazione, più vari i dati
train_std = train_scores.std(axis=1)
test_mean = test_scores.mean(axis=1) #Media e Deviazione su dati di test, più alta la deviazione, più vari i dati
test_std = test_scores.std(axis=1)

plt.figure()
# La Training Accuracy va ad indicare quanto il modello impara bene dai dati di training
plt.plot(*args: train_sizes, train_mean, color='blue', marker='o', markersize=5, label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')
# La Testing Accuracy altro non è che il variare dell'Accuracy man mano che aggiungiamo dati di test
plt.plot(*args: train_sizes, test_mean, color='green', linestyle='--', marker='s', markersize=5, label='Testing Accuracy')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15, color='green')
plt.xlabel('Numero di dati Training')
plt.ylabel('Accuracy')
plt.title(f'Curva Apprendimento - {name}')
plt.legend(loc='lower right')
plt.show()
```

```
# Creazione Immagine della Curva ROC
if hasattr(model, "predict_proba"):
    # probas ha il numero di casi in cui è stato predetto 1
    probas = model.predict_proba(X_test)[: , 1]
    # Calcolo della curva ROC
    fpr, tpr, thresholds = roc_curve(y_test, probas)
    # Calcolo dell'area sotto la curva
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(*args: fpr, tpr, color='darkorange', lw=2, label=f'ROC (Area Sotto La Curva = {roc_auc:.2f})')
    plt.plot(*args: [0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('Falsi Positivi')
    plt.ylabel('Veri Positivi')
    plt.title(f'ROC - {name}')
    plt.legend(loc='lower right')
    plt.show()
```



Ricordiamo cosa indicano dalle definizioni:
“La Learning Curve mostra tipicamente come l'errore (o l'accuratezza) del modello cambia sia sul set di addestramento che su quello di test al variare della dimensione del set di addestramento. Solitamente, si considera la dimensione del set di addestramento lungo l'asse x e l'errore (o l'accuratezza) lungo l'asse y. Idealmente, le curve di addestramento e di test dovrebbero rimanere vicine l'una all'altra. Ciò suggerisce che il modello sta generalizzando bene su dati non visti, poiché le prestazioni sul set di test sono simili a quelle sul set di addestramento.”

e

“La curva ROC mostra il trade-off tra TruePositiveRate e FalsePositiveRate per diverse soglie di decisione del modello. Un modello con una buona capacità discriminante avrà una curva ROC che si avvicina il più possibile all'angolo in alto a sinistra del grafico, indicando un'alta TPR e un basso FPR.”

Detto questo...sono a dir poco entrambe disastrose per il nostro obiettivo, non ci basta determinare che il Random Forest è il modello migliore se ha un'AUC di 0.69, dobbiamo fare qualcosa sul dataset in sé. Andiamo ad esaminarlo meglio...

7 Dataset nel Dettaglio

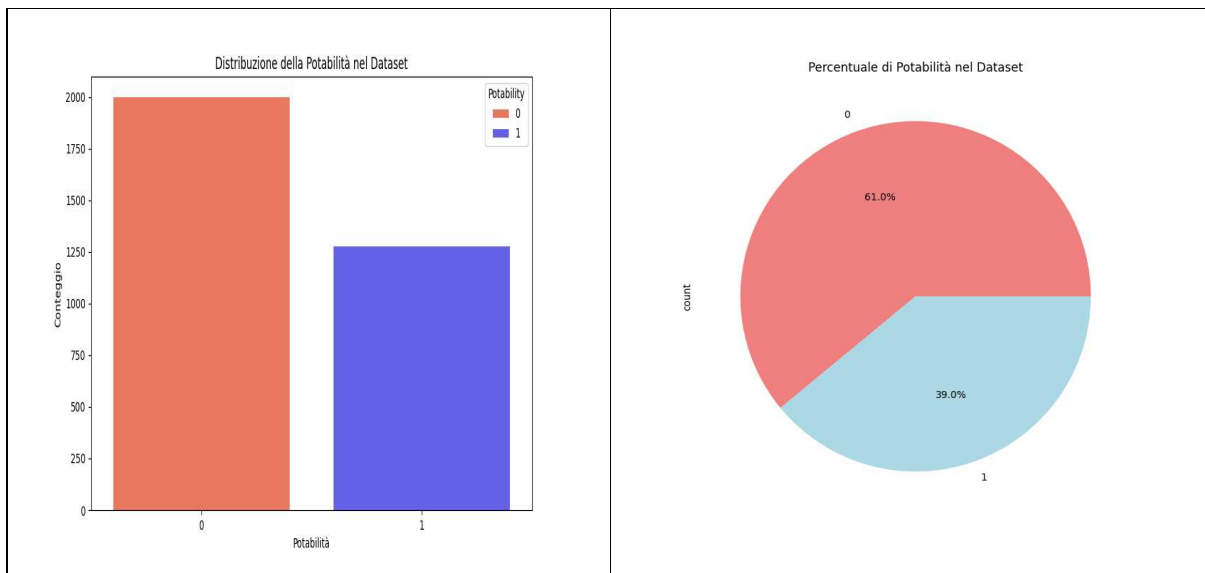
Qui giungiamo alla parte problematica che ho menzionato all’inizio, prima di tutto usiamo questo script (script-visualizza-dataset) per vedere l’effettiva distribuzione in percentuale e in quantità della potabilità, idealmente saranno due valori vicini

```
# Carica il dataset
dataset = pd.read_csv("water_potability.csv")

# Creazione dell'istogramma con conteggio delle potabilità
plt.figure(figsize=(10, 6))
sns.countplot(x='Potability', data=dataset, hue='Potability', palette={0: "#ff6b4a", 1: "#534aff"}, dodge=False)
plt.title('Distribuzione della Potabilità nel Dataset')
plt.xlabel('Potabilità')
plt.ylabel('Conteggio')
plt.show()

# Creazione del grafico a torta
plt.figure(figsize=(8, 8))
dataset['Potability'].value_counts().plot.pie(autopct='%1.1f%%', colors=['lightcoral', 'lightblue'])
plt.title('Percentuale di Potabilità nel Dataset')
plt.show()
```

Il risultato:



Quindi, abbiamo quasi 2000 entry “Non Potabili” che rappresentano il 61% del totale nel dataset, mentre abbiamo circa 1250 entry “Potabili”, che invece ne rappresentano il restante 39%. C’è uno sbilanciamento abbastanza evidente ma prima di pensarci meglio, andiamo a vedere come sono distribuiti i dati all’interno delle entry, per questo useremo quest’ altro script (script-distribuzione):

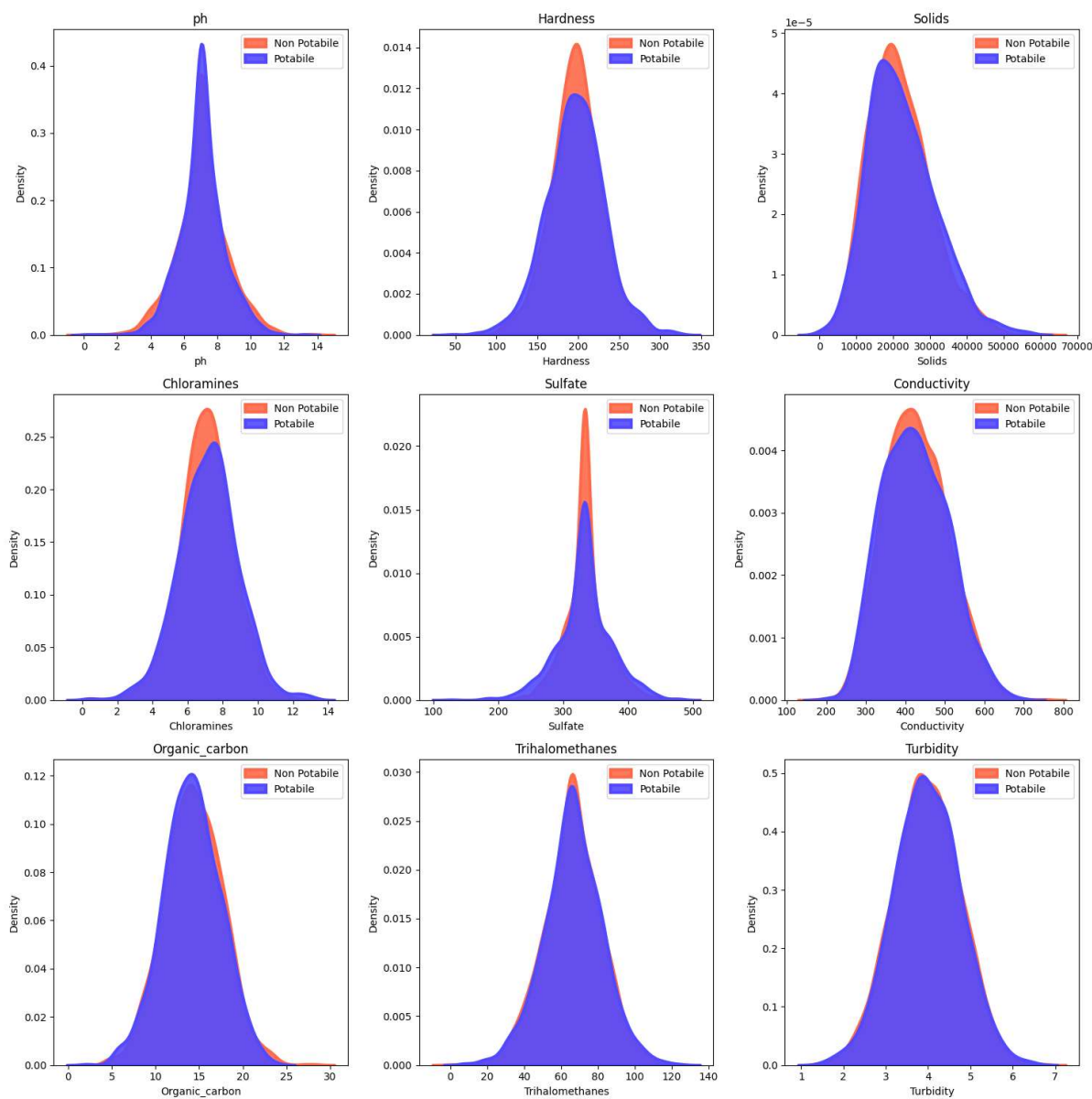
```
# Carica il dataset
dataset = pd.read_csv("water_potability.csv")

# Di questi sceglierne solo uno! Il primo cancella le entry con dati mancanti, la seconda invece fa l'imputazione con media
# dataset = dataset.dropna()
dataset.fillna(dataset.mean(), inplace=True)

# Separazione delle entry potabili e non potabili
non_potabile = dataset.query("Potability == 0")
potabile = dataset.query("Potability == 1")

# Creazione del plot che conterrà tutte le distribuzioni
plt.figure(figsize=(15, 15))
for ax, col in enumerate(dataset.columns[:-1]):
    plt.subplot(*args: 3, 3, ax + 1)
    plt.title(col)
    plotting = sns.kdeplot(x=non_potabile[col], label="Non Potabile", fill=True, common_norm=False, color="#ff6b4a", alpha=0.9, linewidth=3)
    plotting = sns.kdeplot(x=potabile[col], label="Potabile", fill=True, common_norm=False, color="#534aff", alpha=0.9, linewidth=3)
    plt.legend()

plt.tight_layout()
plt.suptitle('Distribuzione delle Features', y=1.08, size=26, weight='bold')
plt.show()
```



Da questa esecuzione deduciamo una notizia buona ed una cattiva, la buona è che per fortuna non c'è un grosso divario tra i valori che portano ad una classificazione “Potabile” o ad una “Non Potabile”.

La cattiva è che ci sono dei valori assolutamente impossibili, per esempio, abbiamo chiaramente dei valori “ph” tremendamente bassi o alti ma comunque classificati come “Potabile”; Questo è un serio problema dato che questo valore (insieme ad alcuni di “Durezza”, “Carbonio”, “Triometani” e “Torbidity”) indica a prescindere che l'acqua non è potabile, indifferentemente dagli altri.

8 Modifiche al Dataset

Dunque, è chiaro che questo dataset non è proprio ottimale per il nostro obiettivo, cioè quello di avere un modello molto preciso, anzi direi proprio che è poco realistico!

Però cercando online per altri dataset non ho avuto fortuna nel trovarne uno che soddisfi le nostre esigenze, quindi sono giunto ad un'idea: “E se il dataset lo aggiustassi io?” online non si trovano versioni realistiche di questo dataset, né tantomeno altre analisi di questo dataset che sono giunte alla mia stessa conclusione, quindi, per cominciare:

Il primo script che andremo a guardare è “script-correzione-dataset”:

```
# Carica il dataset e imputalo con media per evitare che alcune tuple si "salvino"
dataset = pd.read_csv("water_potability.csv")
dataset.fillna(dataset.mean(), inplace=True)

# Lista di valori ph specifici da correggere
ph_errore = [1, 2, 3, 4, 5, 10, 11, 12, 13, 14]

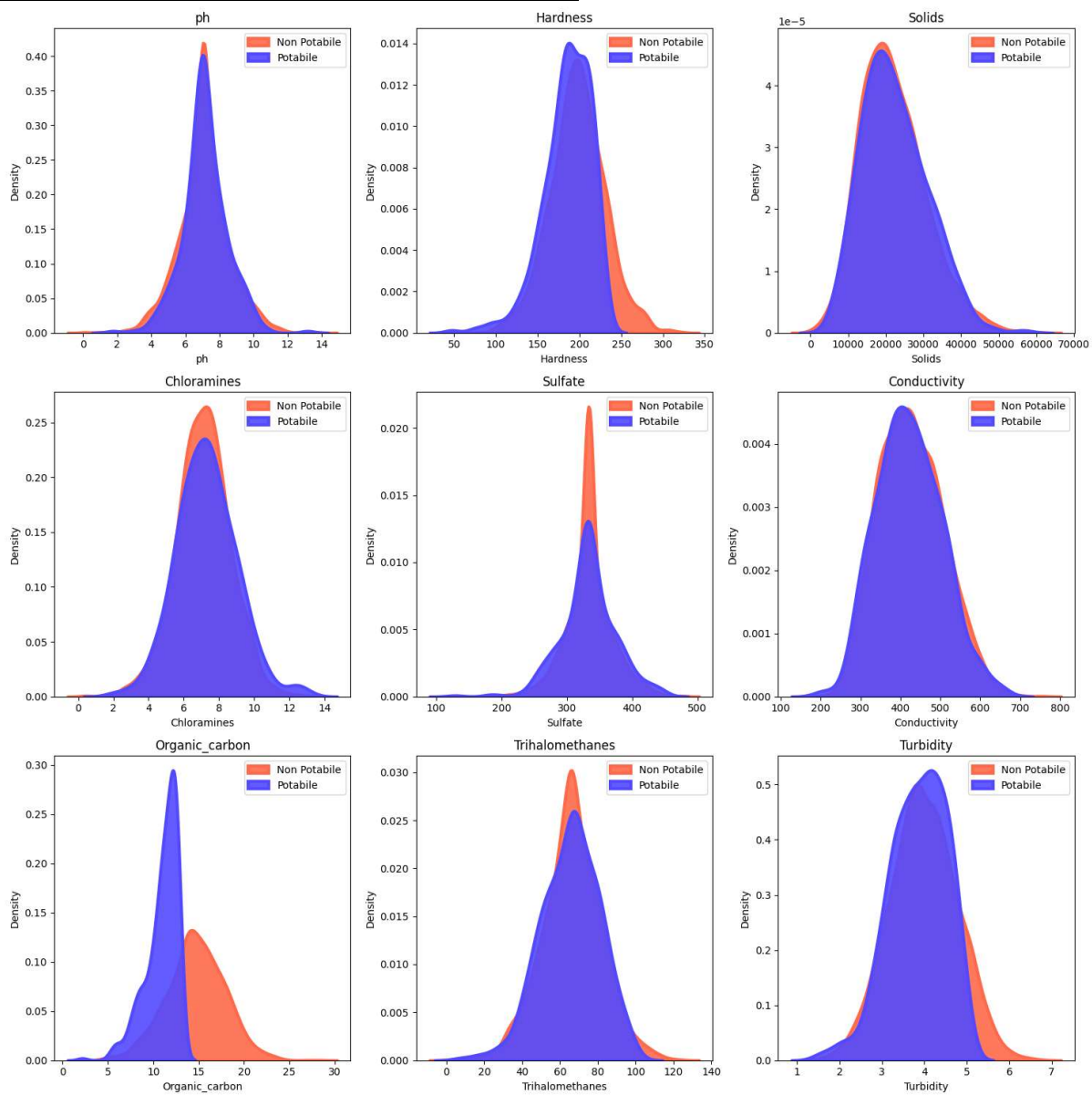
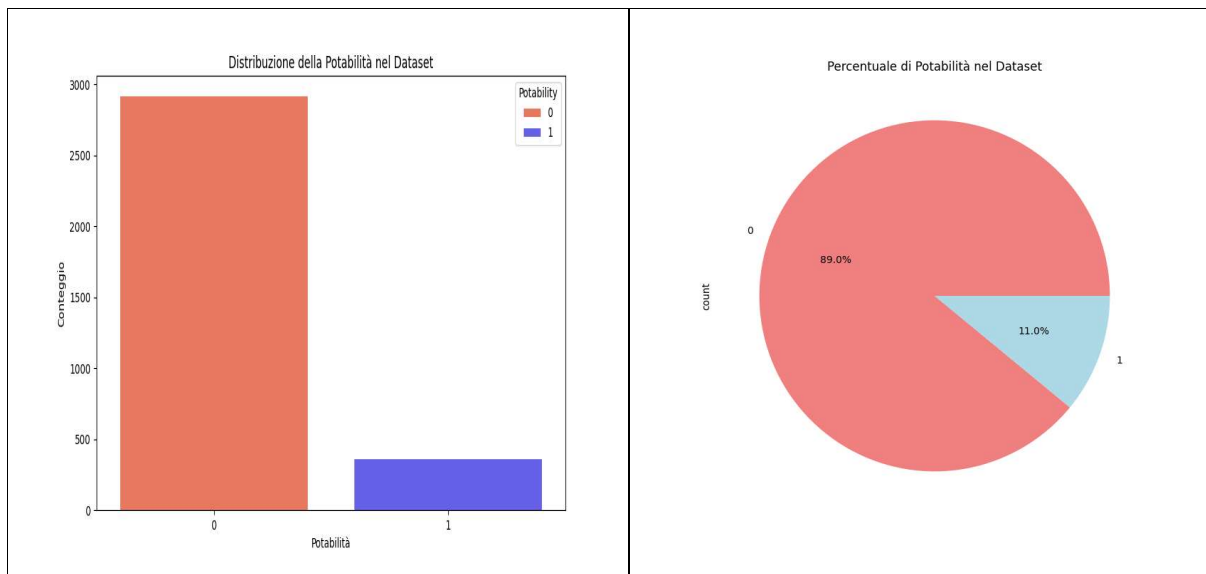
# Individua le entry che hanno valori "impossibili"
error_ph = dataset[(dataset["ph"].isin(ph_errore)) & (dataset["Potability"] == 1)].index
error_hardness = dataset[(dataset["Hardness"] > 230) & (dataset["Potability"] == 1)].index
error_carbon = dataset[(dataset["Organic_carbon"] > 13) & (dataset["Potability"] == 1)].index
error_trihalomethanes = dataset[(dataset["Trihalomethanes"] > 100) & (dataset["Potability"] == 1)].index
error_turbidity = dataset[(dataset["Turbidity"] > 5) & (dataset["Potability"] == 1)].index

# Correggi gli errori impostando la classe a "Non Potabile" per gli indici identificati
dataset.loc[error_ph, "Potability"] = 0
dataset.loc[error_hardness, "Potability"] = 0
dataset.loc[error_carbon, "Potability"] = 0
dataset.loc[error_trihalomethanes, "Potability"] = 0
dataset.loc[error_turbidity, "Potability"] = 0

# Salva il dataset corretto
dataset.to_csv(path_or_buf="water_potability_corrected.csv", index=False)
```

Esso prende il nostro dataset in input, definisce i valori di “ph” che sono da considerare “Non Potabile”, successivamente andrà ad individuare le entry che rientrano nei requisiti di non potabilità per i valori di “ph”, “Durezza”, “Carbonio”, “Trihalometani” e “Torbidity” e poi imposterà tutte queste entry a “Non Potabile” e salverà il dataset in un secondo file csv, convenientemente chiamato “water_potability_corrected.csv”

Adesso andiamo ad esaminare la distribuzione delle entry e dei valori in questo dataset:



È chiaro che questo dataset NON è utilizzabile in questo stato, data la disparità 89% e 11% tra “Non Potabile” e “Potabile”, ma almeno la distribuzione delle feature è diventata più realistica. L’addestramento dei modelli con questo dataset sarà complesso in quanto c’è un chiaro sotto campionamento dal lato “Potabile”, quindi il prossimo passo sarà la Data Augmentation, andremo a creare dati “fantoccio” utilizzando SMOTE (Synthetic Minority Over-sampling Technique), in questo script (script-genera-campioni) e salviamo poi il nuovo dataset col nome di “water_potability_corrected_extended”:

```
# Carica il dataset
dataset = pd.read_csv("water_potability_corrected.csv")

# Di questi sceglierne solo uno! Il primo cancella le entry con dati mancanti, la seconda invece fa l'imputazione con la media
# dataset = dataset.dropna()
dataset.fillna(dataset.mean(), inplace=True)

# Separazione di features X (tutti i valori delle feature tranne Potability)
# dal target y (cioè la "Potability", il valore da predire)
X = dataset.drop(labels="Potability", axis=1)
y = dataset["Potability"]

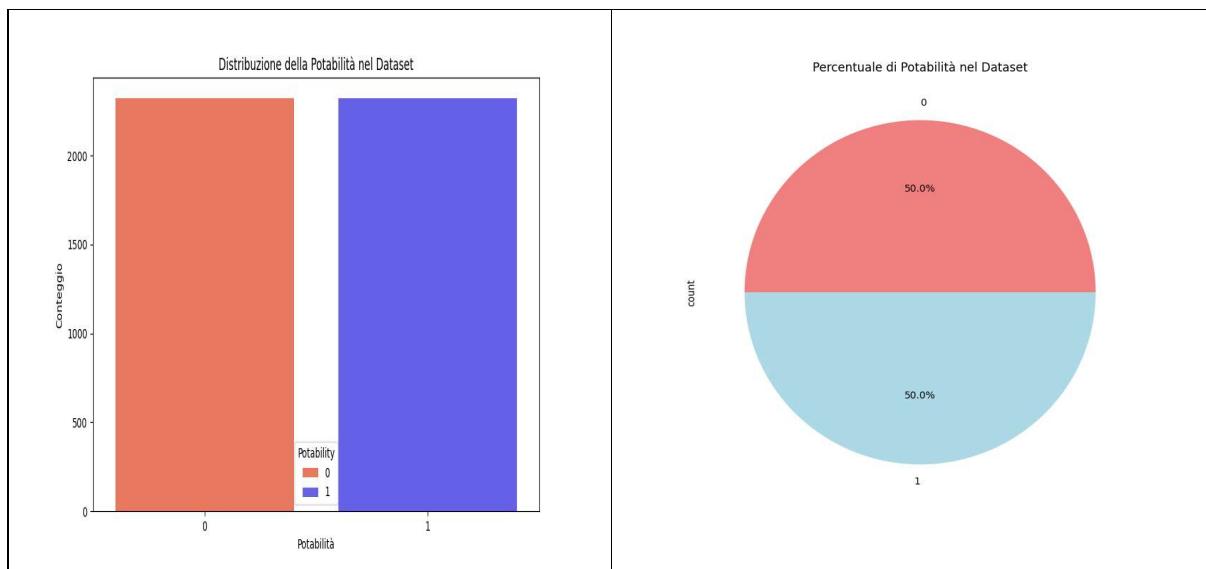
# Suddivisione del dataset in training e testing, SMOTE ne ha bisogno
X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.2, random_state=42)

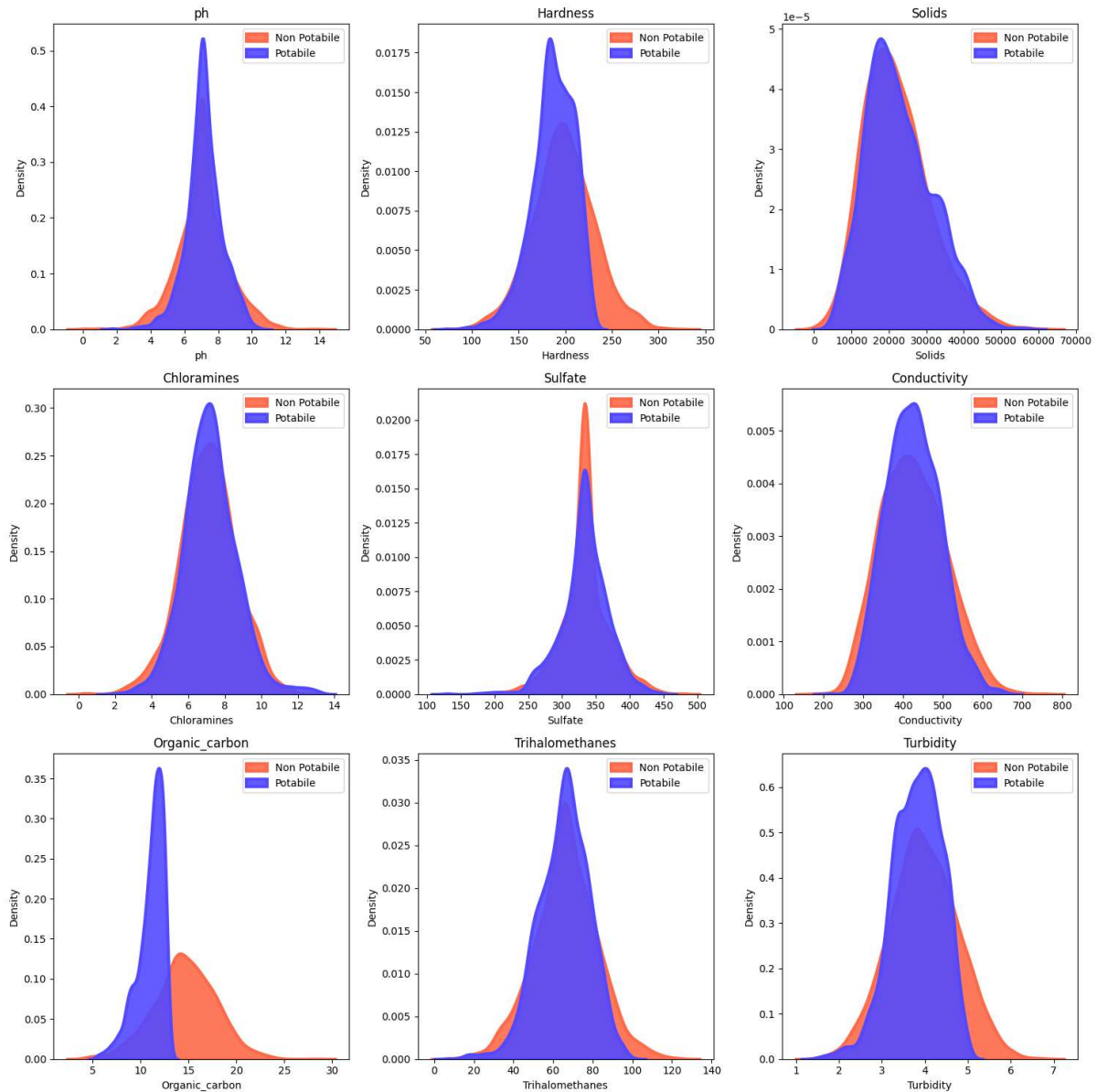
# Utilizzo di SMOTE per i campioni sintetici
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Copia del dataset originale
balanced_dataset = dataset.copy()

# Salvataggio del nuovo dataset, X_resampled ha i valori, y_resampled ha la Potabilità
balanced_dataset = pd.DataFrame(X_resampled, columns=X.columns)
balanced_dataset['Potability'] = y_resampled
balanced_dataset.to_csv(path_or_buf="water_potability_corrected_extended.csv", index=False)
```

Una volta fatto questo, andiamo di nuovo ad esaminare la distribuzione delle entry e dei valori:





Dunque, 50% e 50% su “Potabile” e “Non Potabile”, e la distribuzione dei valori ha subito leggere variazioni, ma è apposto sul fattore realismo, dunque, andiamo ad usare questo nuovo dataset su ogni modello usato precedentemente!

9 Data Training, Risultati e Confronti finali

I risultati che ora mostro sono stati copiati dalla console di esecuzione dello script:

Metriche di Decision Tree:

- Accuracy: 0.920344456
- Precision: 0.898924731
- Recall: 0.939325843
- F1-Score: 0.918681319
- Confusion Matrix:

```
[[437 47]
```

```
[ 27 418]]
```

Metriche di Random Forest:

- Accuracy: 0.930032293
- Precision: 0.884615385
- Recall: 0.982022472
- F1-Score: 0.930777423
- Confusion Matrix:

```
[[427 57]
```

```
[ 8 437]]
```

Metriche di Logistic Regression:

- Accuracy: 0.780409042
- Precision: 0.751565762
- Recall: 0.808988764
- F1-Score: 0.779220779
- Confusion Matrix:

```
[[365 119]
```

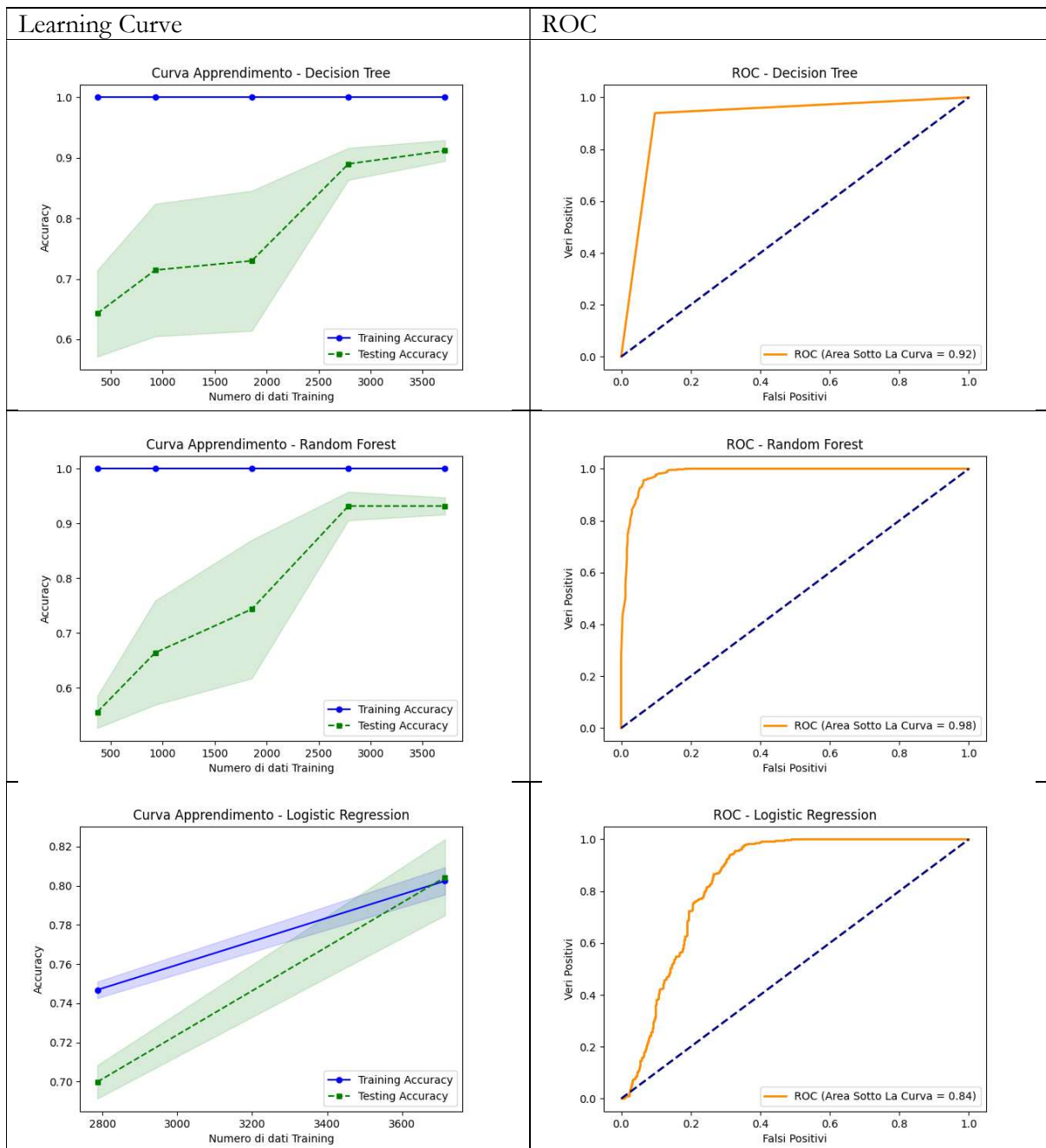
```
[ 85 360]]
```

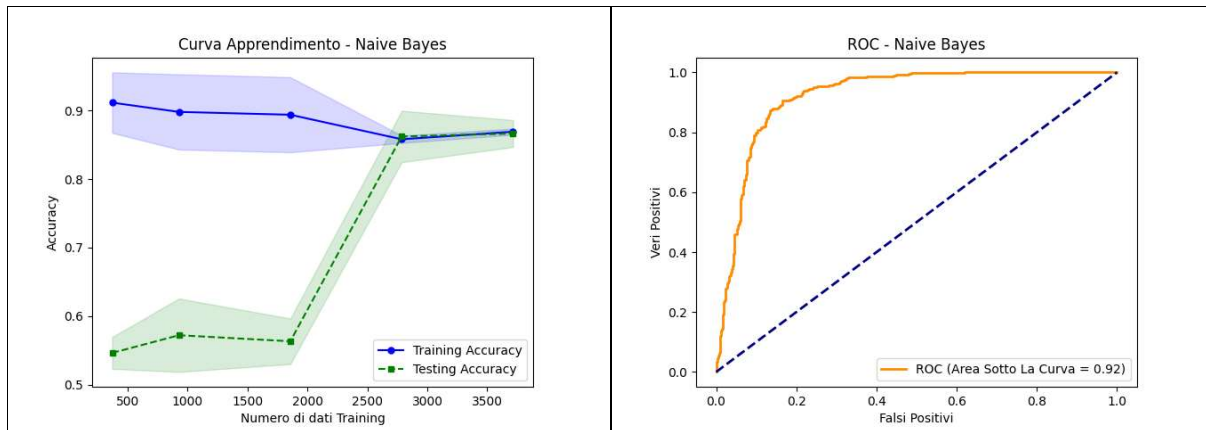
Metriche di Naive Bayes:

- Accuracy: 0.862217438
- Precision: 0.824130879
- Recall: 0.905617978
- F1-Score: 0.862955032
- Confusion Matrix:

```
[[398 86]
```

```
[ 42 403]]
```





Adesso i risultati sono davvero soddisfacenti! Il modello migliore rimane Random Forest, con un Accuracy di ben 93%, la sua curva ROC arriva ad un pelo dal punteggio pieno, per la Learning Curve possiamo però vedere come il modello stia imparando a memoria troppo in fretta, ma ciò non l'ha fermato dall'avere una Validation Accuracy (Testing Accuracy sull'immagine) in crescita costante.

La seconda scelta migliore è il Decision Tree, in termini di Accuracy, con il suo 92%. Una curva ROC che si taglia "a gomito" e una Learning Curve molto simile a quella del Random Forest.

Naive Bayes è al terzo posto con l'86% di Accuracy, una curva ROC dal valore AUC identico a quello del Decision Tree ed una Learning Curve dove verso l'80% di tutti i sample esaminati le due linee si incrociano, che magari sia già quello il suo massimo potenziale? Dopotutto non si scolleranno più.

All'ultimo posto troviamo Logistic Regression, 78% di Accuracy, curva ROC più bassa di tutte quelle a disposizione (84) e la Learning Curve letteralmente lineare, le due linee si incrociano alla fine senza "scatti" man mano che passa il tempo.

10 Interattività

Per chiudere in bellezza, ho creato anche un ultimo script, script-interattivo, per dare la possibilità all'utente di inserire in input delle proprie entry di cui far predire il valore al modello, dopotutto, era questo lo scopo del progetto, creare un modello capace di classificare la potabilità dell'acqua, tramite dati inseriti dall'utente, con un minimo tasso di errore.

```
# Carica il dataset
dataset = pd.read_csv("water_potability_corrected_extended.csv")
#Stampa delle linee guida su console per specificare all'utente che dati indicano cosa
print("Linee guida:\nph: 0-14\nHardness: Calcio e Magnesio nell'acqua in mg/L\nSolids: Solidi Disciolti")

# Imputazione tramite Media (mean) in caso ci dovessero essere problemi in lettura
dataset.fillna(dataset.mean(), inplace=True)

# Separazione delle features X (tutti i valori delle feature tranne Potability)
# dal target y (cioè la "Potability", il valore da predire)
X = dataset.drop(labels="Potability", axis=1)
y = dataset["Potability"]

# Creiamoci il RandomForest...
randomforest = RandomForestClassifier(random_state=42)
randomforest.fit(X, y)

# E qui la variabile per continuare
continua = True
```

```
while continua:
    # Chiedi all'utente di inserire i valori per una nuova entry (tranne "Potability")
    new_entry = {}
    for column in X.columns:
        value = input(f"Inserisci il valore per '{column}': ")
        new_entry[column] = float(value)

    # Creazione di un DataFrame con la nuova entry
    new_entry_df = pd.DataFrame([new_entry])

    # Effettua la predizione sul nuovo dato
    prediction = randomforest.predict(new_entry_df)

    # Stampa il risultato
    if prediction[0] != 1:
        print("L'acqua non è potabile.")
    else:
        print("L'acqua è potabile.")

    ris = input(f"Vuoi inserire un'altra entry? (y/n): ")
    if ris.lower() != 'y':
        continua = False
```

Esecuzione:

```
C:\Users\nunzi\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Users\nunzi\Desktop\Acqua\DatasetCorretto\script-interattivo.py
Linee guida:
ph: 0-14
Hardness: Calcio e Magnesio nell'acqua in mg/L
Solids: Solidi Disciolti nell'acqua in ppm
Chloramines: Cloramina (un agente disinfettante) nell'acqua in ppm
Sulfate: Solfato nell'acqua in mg/L
Conductivity: Capacità di condurre elettricità nell'acqua in microsiemens per centimetro (µS/cm)
Organic_Carbon: Il carbonio nell'acqua in ppm
Trihalomethanes: Trialometani (sottoprodotti della disinfezione) nell'acqua in microgrammi per litro (µg/L)
Turbidity: Turbidità (la capacità dell'acqua di diffondere la luce) in Unità Nefelometriche di Turbidità (NTU)

Inserisci il valore per 'ph': |
```

```
Inserisci il valore per 'ph': 7
Inserisci il valore per 'Hardness': 300
Inserisci il valore per 'Solids': 200
Inserisci il valore per 'Chloramines': 6
Inserisci il valore per 'Sulfate': 125
Inserisci il valore per 'Conductivity': 1000
Inserisci il valore per 'Organic_carbon': 11
Inserisci il valore per 'Trihalomethanes': 70
Inserisci il valore per 'Turbidity': 3
L'acqua non è potabile.
Vuoi inserire un'altra entry? (y/n): n

Process finished with exit code 0
|
```

Ha ragione? Sì!

“La durezza dell’acqua è generalmente espressa come mg/L di CaCO_3 . La classificazione della durezza dell’acqua in base alla concentrazione di CaCO_3 è la seguente¹²:

Fino a 70 mg/L: molto dolce

Da 70 a 140 mg/L: dolce

Da 140 a 210 mg/L: mediamente dura

Da 210 a 320 mg/L: discretamente dura

Da 320 a 540 mg/L: dura

Oltre 540 mg/L: molto dura”

300mg/L è sicuramente non potabile!