Matteo Chinellato

matricola: 0322500038

matteo.chinellato_4939@studenti.uniroma5.it

Progetto da consegnare per l'esame di Apprendimento Automatico e Apprendimento Profondo

Prof.ssa Scarpato

25 gennaio 2026

Scelta del dataset nel sito UCI Dataset

Il dataset che ho scelto per l'esercitazione è *Early Stage Diabetes Risk Prediction*:
https://archive.ics.uci.edu/dataset/529/early+stage+diabetes+risk+prediction+dataset

La scelta del dataset è stata orientata sul dominio attinente al mio ambito professionale, ovvero la sanità. Questa caratteristica non ha alcuna implicazione sulle finalità dell'esercitazione pertanto non è stata considerata un prerequisito ma un valore aggiunto per finalità e piacere personale.

Le valutazioni effettuate, al fine di consentire una buona riuscita dell'esercitazione, sono state:

a) **Numero di feature.** 16 feature sono un buon compromesso per il livello di complessità e per giustificare la riduzione delle dimensionalità senza perdere troppa varianza spiegata.
b) **Tipo di feature.** La grande prevalenza di feature binarie, 15 su 16, è ideale per algoritmi di classificazione come *SVM* e *Logistic Regression* che lavorano bene con confini di decisione netti. La presenza della feature intera ed ordinale, l'età, motiva la necessità di standardizzazione per evitare la sua predominanza sulle altre. Le feature, infine, si prestano bene per l'*analisi SHAP* al fine di determinare il contributo di ogni sintomo clinico (feature) per la previsione della diagnosi della malattia (classe).
c) **Tipo di target.** Un target binario (diagnosi malattia: positivo/negativo) permette di ottimizzare il modello sulla metrica della *recall* e si presta bene anch'esso per l'*analisi SHAP*. La suddivisione delle istanze in due sole classi garantisce inoltre al modello un numero sufficiente di esempi.
d) **Numero di istanze.** Le dimensioni del dataset, 520 istanze, si prestano ottimamente all'applicazione di algoritmi di *Ensemble Learning* come il *Random Forest*. La natura non eccessivamente onerosa del campione permette tempi di addestramento pressoché istantanei, facilitando la reiterazione veloce delle fasi di test. Questo aspetto è fondamentale ai fini dell'esercitazione, ovvero affinare il modello e approfondire la comprensione delle dinamiche tra le variabili cliniche.

Pre-processing dei dati

Il dataset presenta esclusivamente valori di tipo stringa ed è stato pertanto necessario trasformarli in valori numerici; considerando che le feature sono di tipo binario (*Yes/No* e *Male/Female*) ho utilizzato la tecnica del *Label Encoding* evitando di aumentare il numero di colonne e preservando quindi la compattezza del dataset.

```
# utilizzo Label Encoding per la codifica delle variabili categoriche (ecetto l'età sono tutte testuali ma binarie)
df_encoded = df.copy()
le = LabelEncoder()
for col in df_encoded.columns:
    if df_encoded[col].dtype == 'object':
        df_encoded[col] = le.fit_transform(df_encoded[col])
```

Avendo un numero contenuto di istanze, pari a 520, ho suddiviso il dataset in *Traning*, *Validation* e *Test set* con le proporzioni 70%, 15% e 15%. Tale rapporto è un compromesso che garantisce un volume di dati sufficienti per l'addestramento; al fine di garantire la proporzione delle classi trai tre set ho utilizzato la stratificazione.

Anche se non strettamente necessario ho deciso, ai fini dell'esercitazione, di utilizzare anche il *Validation set* come verifica intermedia sulla consistenza delle performance.

```
# suddivido il dataset in training (70%), validation (15%) e test (15%)
# le percentuali scelte sono dovute al numerto limitato di righe del dataset
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.50, random_state=42, stratify=y_temp)
```

Per prevenire che l'età, l'unica variabile intera e ordinata, eserciti una dominanza sulle altre di tipo binario, ho utilizzato il metodo *StandardScaler*. Ho quindi effettuato la standardizzazione:

- sia di *Traning*, *Validation* e *Test set* separatamente evitando contaminazioni statistiche, *Data Leakage*, per simulare correttamente l'applicazione del modello su nuovi pazienti;
- sia dell'intero dataset per consentire, con la visualizzazione globale tramite *PCA*, di proiettare tutti i campioni nello stesso spazio bidimensionale rendendo visibile la struttura naturale delle classi di pazienti negativi e positivi.

```
# effettuo lo scaling per evitare che la feature età domini le altre
scaler = StandardScaler()
# effettuo lo scaling speratamente su Traning, Validation e Test set
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
# effettuo anche lo scaling globale per la visualizzazione PCA
X_scaled_full = scaler.fit_transform(X)
```

Analisi delle feature del dataset scelto mediante l'implementazione dell'algoritmo PCA per le feature scelte e visualizzazione dei risultati

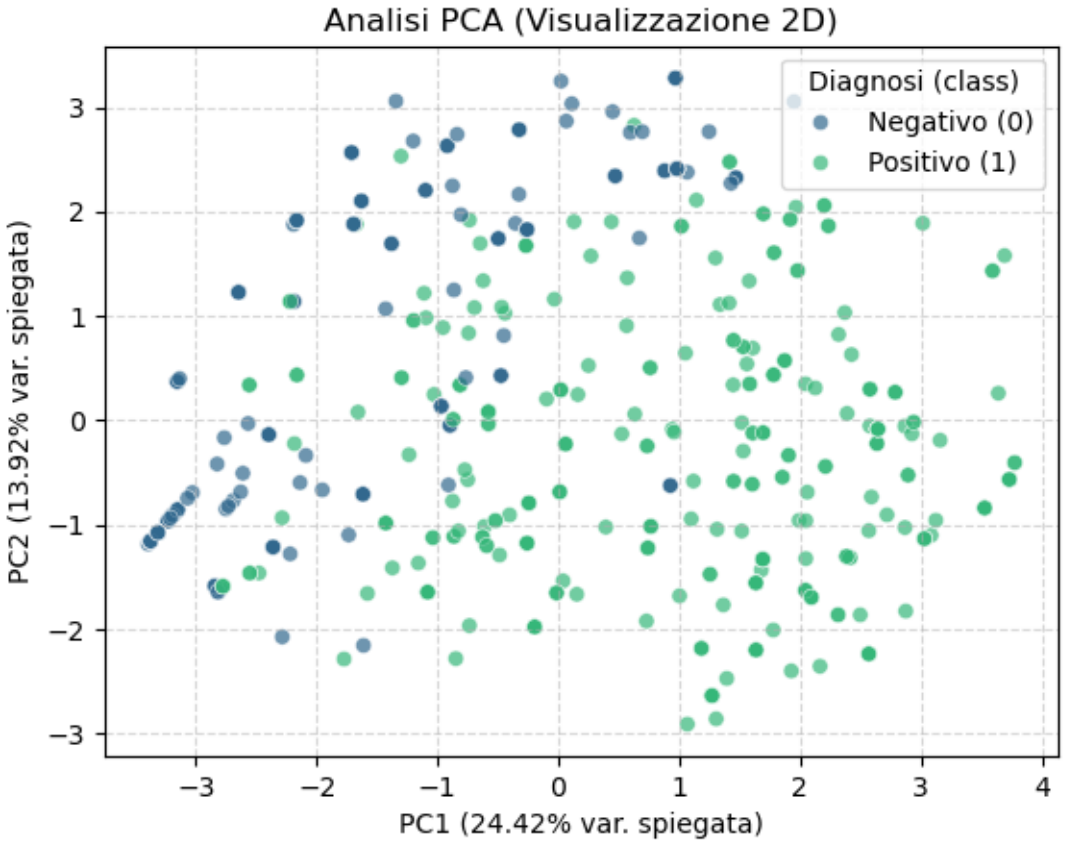
Valutato il numero ridotto di feature che caratterizzano il dataset, 16 totali, ho ritenuto non necessaria la riduzione del dataset per l'addestramento dei modelli. Ho quindi applicato l'algoritmo *PCA* sulla totalità delle feature, per le sole finalità di analisi esplorativa, permettendo di estrarre la massima varianza possibile catturabile da due sole componenti e di rappresentarla sul piano bidimensionale. Ho infine addestrato i modelli sulle feature originali per preservare l'integrità informativa e l'interpretabilità clinica dei risultati.

```
# ------------------------------------------------------------------------
# 1. Analisi delle feature del dataset scelto mediante l'implementazione
# dell'algoritmo PCA per le feature scelte e visualizzazione dei risultati
# ------------------------------------------------------------------------
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled_full)
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['class'] = y.values

#plt.figure(figsize=(10, 6))
# creo il grafico
ax = sns.scatterplot(x='PC1', y='PC2', hue='class', data=pca_df, palette='viridis', alpha=0.7)
# modifico la legenda
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles=handles, labels=['Negativo (0)', 'Positivo (1)'], title='Diagnosi (class)')
# modifico titolo e nome degli assi
plt.title('Analisi PCA (Visualizzazione 2D)')
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.2%} var. spiegata)')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.2%} var. spiegata)')
plt.grid(True, linestyle='--', alpha=0.5)
plt.savefig('analisi_pca.png')
plt.show()
```

La scelta effettuata ha permesso di valutare visivamente come le prime due componenti principali riescano a rappresentare quasi il 40% delle varianza totale del dataset e ad identificare le due classi di pazienti (negativi e positivi).

La PC1 è la componente più importante e spiega da sola circa un quarto della variabilità totale; nel piano bidimensionale è rappresentato in modo evidente il cambiamento della diagnosi dei pazienti spostandosi da sinistra verso destra lungo l'asse orizzontale. La PC2 ha invece una varianza poco superiore alla metà di PC1; si nota comunque come i pazienti negativi tendando a concentrarsi maggiormente nella parte superiore sinistra del grafico.

In sintesi, la visualizzazione *PCA* applicata a tutte le feature conferma la validità del dataset. La proiezione nel piano bidimensionale dimostra che i sintomi sono discriminatori: le due classi di pazienti non sono infatti completamente mescolate tra loro, si osserva una netta polarizzazione dei pazienti positivi verso il quadrante destro contrapposta a una concentrazione dei pazienti negativi sulla sinistra, in particolar modo nella parte alta, sebbene nell'area centrale persista una zona di sovrapposizione tra le due classi.


Analisi PCA (Visualizzazione 2D)
Diagnosi (class)
Negativo (0)
Positivo (1)
PC2 (13.92% var. spiegata)
PC1 (24.42% var. spiegata)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4

Implementazione di almeno tre degli algoritmi di classificazione affrontati nel corso

L'analisi *PCA* effettuata giustifica l'adozione di modelli di classificazione non lineari, capaci di mappare correttamente anche la zona di parziale sovrapposizione rilevata nell'area centrale del piano.

Per completare l'esercitazione ho deciso di implementare i seguenti tre algoritmi:

1. ***Logistic Regression***. Rappresenta il modello lineare per eccellenza e assume l'esistenza di una separazione netta e rettilinea tra le classi. Sebbene l'analisi *PCA* suggerisca che non sia la soluzione più performante per questo dataset a causa della zona di sovrapposizione, ho scelto di includerlo nella esercitazione come benchmark per valutare l'effettivo miglioramento delle performance garantito dagli algoritmi non lineari.
2. ***SVM (Support Vector Machines)***. Specifico per la gestione delle zone di ambiguità, questo algoritmo si concentra sui casi critici vicini al confine di decisione (vettori di supporto). La sua capacità di trovare l'iperpiano che massimizza il margine tra le classi lo rende un candidato ideale per risolvere le sovrapposizioni evidenziate dalla *PCA*.
3. ***Random Forest***. Basato su una foresta di alberi decisionali indipendenti, mitiga il rischio di *overfitting* attraverso la tecnica del *voting*. La sua struttura gerarchica è congeniale alla natura clinica dei dati poiché mappa interazioni complesse tra sintomi che sfuggono ai modelli lineari. Essendo in grado di fornire una gerarchia sull'importanza dei sintomi, rappresenta l'approccio più sofisticato tra quelli scelti e giustifica l'implementazione dell'approccio di spiegabilità *SHAP*.

In fase di addestramento, ho implementato la procedura di *5-Fold Cross-Validation* sul *Training set* per mitigare il rischio di *overfitting* e garantire che la capacità predittiva dei modelli sia indipendente dalla specifica suddivisione dei dati.

```
# ------------------------------------------------------------------------------------
# 2. Implementazione di almeno tre degli algoritmi di classificazione affrontati nel corso
# ------------------------------------------------------------------------------------
# definisco gli algoritmi di classificazione che voglio utilizzare
models = {
    "Logistic Regression": LogisticRegression(random_state=42), # modello lineare, non idoneo
    "SVM": SVC(probability=True, random_state=42), # specifico per le zone di sovrapposizione, idoneo
    "Random Forest": RandomForestClassifier(random_state=42) # struttura ad albero, particolarmente idoneo
}

results_list = [] # preparo la struttura dati per memorarizzare i dati dei tre modelli

# ciclo le operazioni sui tre modelli
for name, model in models.items():

    # eseguo la cross validation su 5 fold
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')
    # addestro i modelli esclusivamente sul Training Set Scalato
    model.fit(X_train_scaled, y_train)
    # verifico se il modello sta imparando bene senza andare in overfitting
    y_val_pred = model.predict(X_val_scaled)
    # chiedo ai modelli addestrati di effettuare la previsione sul Test Set Scalato
    y_pred = model.predict(X_test_scaled)
    # chiedo ai modelli addestrati di determinare la probabilità
    y_proba = model.predict_proba(X_test_scaled)[:, 1]
```

Implementazione del calcolo delle metriche di valutazione precision, recall, f-measure Accuracy, Roc AUC ed implementazione della visualizzazione della matrice di confusione e della ROC AUC Curve

Conclusa la fase di addestramento, ho condotto un'analisi approfondita delle prestazioni dei tre modelli (*Logistic Regression*, *SVM* e *Random Forest*) utilizzando esclusivamente il *Test set*. Per ogni modello ho calcolato le seguenti metriche:

- ***Accuracy***. La percentuale di previsioni corrette sul totale dei casi analizzati.
- ***Precision***. La capacità del modello di minimizzare i falsi positivi, riducendo la possibilità di classificare un paziente sano (negativo) come malato (positivo).
- ***Recall***. La capacità del modello di minimizzare i falsi negativi, riducendo la possibilità di classificare un paziente malato (positivo) come sano (negativo).
- ***F1-Measure***. La media armonica tra *Precision* e *Recall*, che fornisce un indice unico della robustezza del modello e del bilanciamento tra i due tipi di errore.
- ***ROC AUC***. La capacità del modello di distinguere tra le due classi, pazienti sani (negativi) e malati (positivi), al variare della soglia di classificazione.

In coerenza con la scelta progettuale di suddividere il dataset in *Traning*, *Validation* e *Test set*, ho calcolato l'*Accuracy* anche sul *Validation set* per monitorare la stabilità dell'apprendimento.

Infine, ho organizzato i risultati in formato tabellare per facilitarne il confronto e ho prodotto per ogni modello la *Matrice di Confusione* e la *ROC AUC Curve* per un'analisi visiva della capacità discriminante.

```
# ------------------------------------------------------------------------------------------------------------
# 3. Implementazione del calcolo delle metriche di valutazione precision, recall, f-measure, accuracy, Roc AUC
# ------------------------------------------------------------------------------------------------------------
metrics = {
    "Modello": name,
    # Media dell'accuratezza sui 5 fold della cross-validation su training set
    "Accuracy_Cross_Validation": cv_scores.mean(),
     # percentuale di previsioni corrette sul totale dei casi su validation set
    "Accuracy_Validation_set": accuracy_score(y_val, y_val_pred),
     # percentuale di previsioni corrette sul totale dei casi su test set
    "Accuracy_Test_set": accuracy_score(y_test, y_pred),
    # capacità del modello di evitare falsi positivi
    "Precision_Test_set": precision_score(y_test, y_pred),
     # capacità del modello di evitare falsi negativi
    "Recall_Test_set": recall_score(y_test, y_pred),
     # media armonica tra precision e recall
    "F1-measure_Test_set": f1_score(y_test, y_pred),
     # capacità del modello di distinguere le due classi
    "ROC_AUC_Test_set": roc_auc_score(y_test, y_proba)
}
results_list.append(metrics)

# ------------------------------------------------------------------------------------------------
# 4. Implementazione della visualizzazione della matrice di confusione e della ROC AUC Curve
# ------------------------------------------------------------------------------------------------
 # calcolo i dati per disegnare la matrice di confusione
cm = confusion_matrix(y_test, y_pred)
# disegno la matrice di confusione
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Negativo', 'Positivo'])
disp.plot(cmap='Blues')
plt.title(f'{name} - Matrice di Confusione')
plt.xlabel('Classe prevista')
plt.ylabel('Classe effettiva')
plt.savefig(f'matrice_confusione_{name.lower().replace(" ", "_")}.png')
plt.show()
```

```
    # calcolo i dati necessari per disegnare la curva Roc AUC
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    # disegno la curva Roc AUC
    plt.plot([0, 1], [0, 1], color='navy', linestyle='--', label='Random (AUC = 0.50)')
    plt.plot(fpr, tpr, label=f'{name} (AUC = {metrics["ROC_AUC_Test_set"]:.2f})')
    plt.title(f'{name} - Curva Roc AUC')
    plt.xlabel('Tassi Falsi Positivi')
    plt.ylabel('Tasso Falsi Negativi')
    plt.legend()
    plt.grid(alpha=0.3)
    plt.savefig(f'curva_Roc_AUC_{name.lower().replace(" ", "_")}.png')
    plt.show()
# fine ciclo

# stampo in formato tabellare il riepilogo delle metriche rilevate sui modelli evidenziando i risultati migliori
metrics_summary = pd.DataFrame(results_list)
print("\n ----- Confronto riassuntivo Metriche di Valutazione ----- ")
styled_results = metrics_summary.style.highlight_max( #evidenzio il valori massimi per ciascuna colonna
    subset=metrics_summary.columns[1:], # salto la colonna Modello altrimenti mi evidenzia SVM
    color='lightgreen', # colore verde chiaro
    axis=0 # identifica il valore massimo
).hide(axis='index')
display(styled_results)
```

L'analisi approfondita delle prestazioni ha consentito di quantificare l'efficacia diagnostica dei modelli e di visualizzarne la capacità discriminativa.

I risultati ottenuti confermano le previsioni iniziali. *SVM* si è dimostrato un modello solido, ma è stato superato dal *Random Forest*. Quest'ultimo ha raggiunto una *Recall* pari a 1.000 (identificando la totalità dei pazienti positivi sul *Test set*, ovvero zero falsi negativi) e una *ROC AUC* di 0.9965, dimostrando una capacità di distinzione tra le classi quasi perfetta (l'unico errore rilevato sul *Test set* consiste in un falso positivo, scenario preferibile in ambito medico rispetto ad una falso negativo).

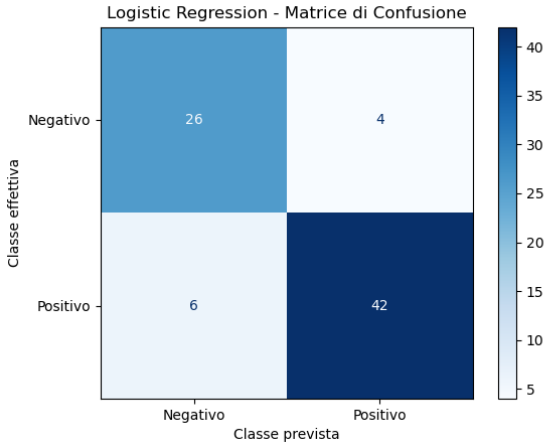
Logistic Regression ha invece mostrato un calo evidente delle performance, confermando come la sua natura lineare sia meno idonea a rappresentare la complessità del dataset scelto per l'esercitazione.

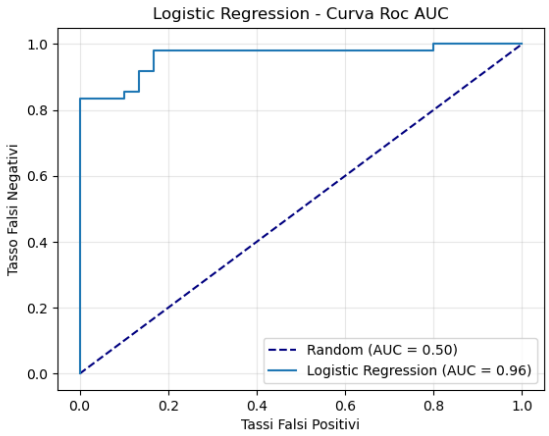
Infine, la coerenza tra i valori di *Accuracy* ottenuti in *Cross-Validation* sul *Training Set*, sul *Validation Set* e sul *Test Set* conferma l'assenza di fenomeni di *overfitting* per i modelli non lineari, *SVM* e *Random Forest*, garantendo l'affidabilità del sistema anche su dati mai visti in precedenza. Al contrario, il delta marcato riscontrato nel modello di *Logistic Regression,* pari a circa il 10% tra fase di addestramento e test, evidenzia una minore robustezza dell'approccio lineare nel generalizzare le dinamiche del dataset.

```
 ----- Confronto riassuntivo Metriche di Valutazione -----
```

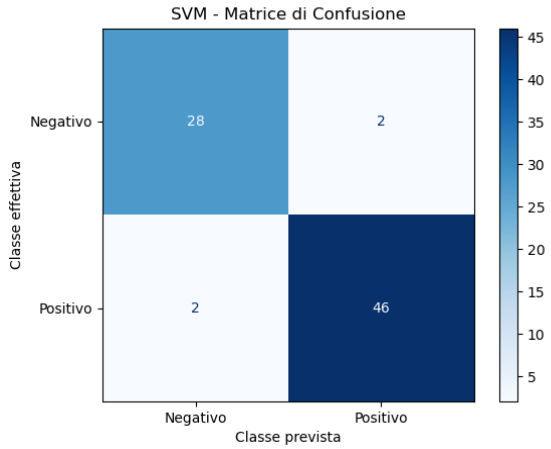
Modello	CV_Accuracy_Mean	Accuracy_Validation_set	Accuracy_Test_set	Precision_Test_set	Recall_Test_set	F1-measure_Test_set	ROC_AUC_Test_set
Logistic Regression	0.922907	0.974359	0.871795	0.913043	0.875000	0.893617	0.962500
SVM	0.961492	0.974359	0.948718	0.958333	0.958333	0.958333	0.993750
Random Forest	0.967047	0.987179	0.987179	0.979592	1.000000	0.989691	0.996528

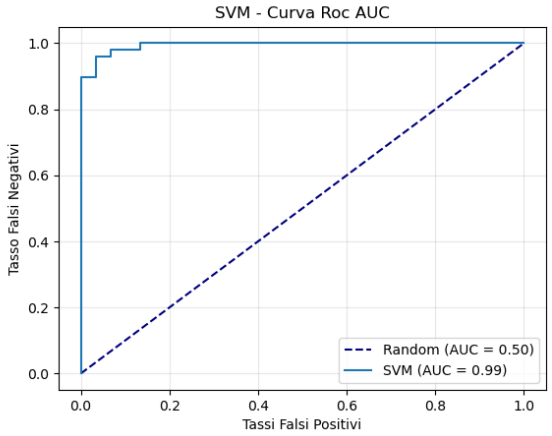
LOGISTIC REGRESSION



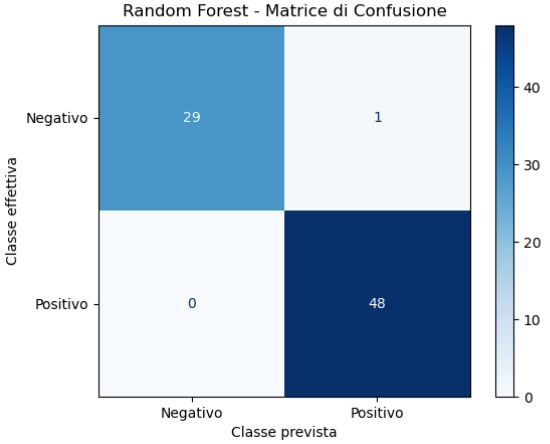


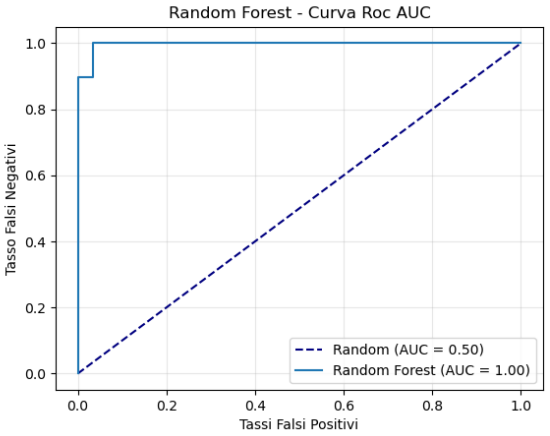
SVM – SUPPORT VECTOR MACHINES





RANDOM FOREST





Implementazione dell’approccio di spiegabilità SHAP

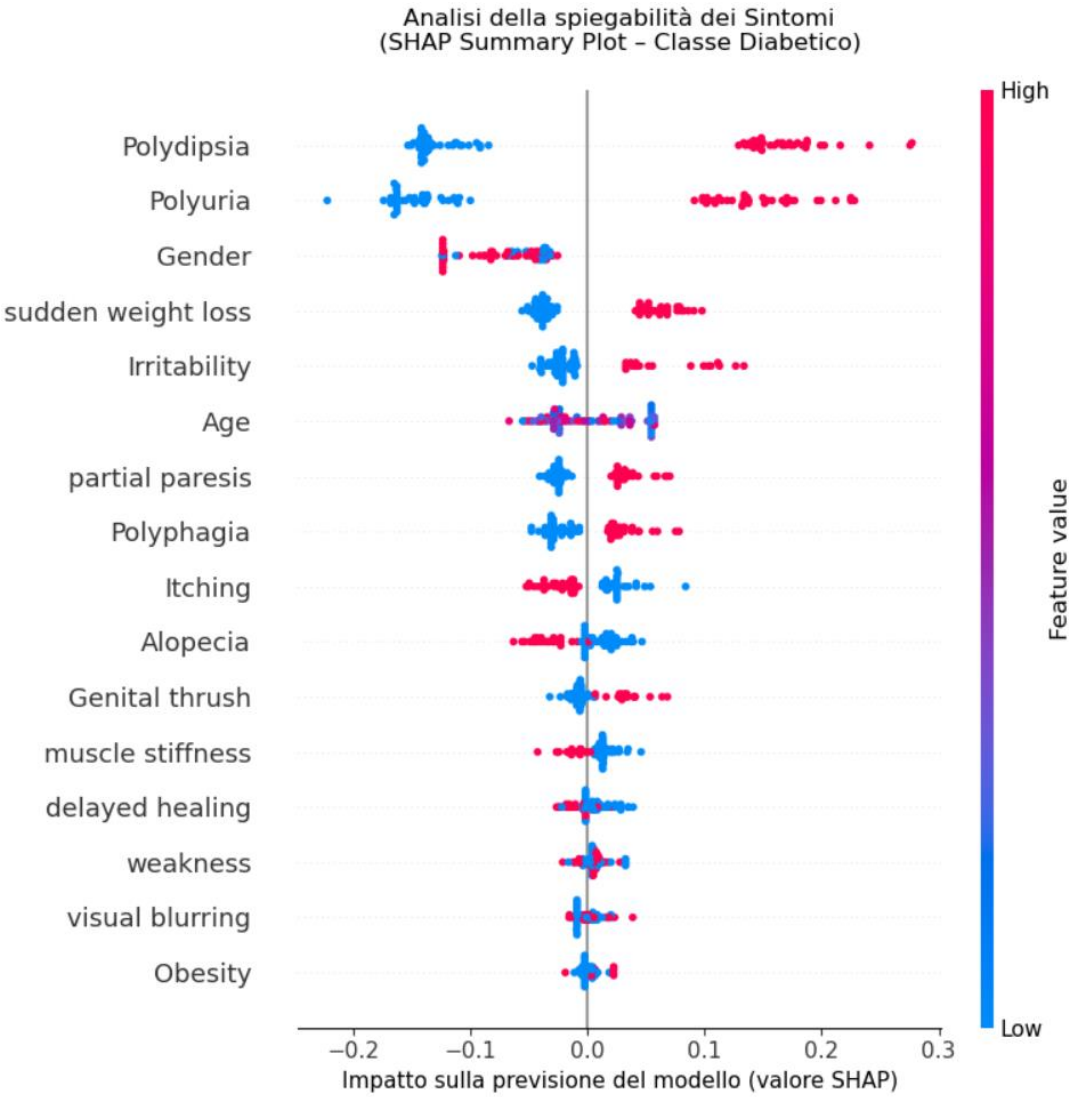
Dato il dominio medico dell’esercitazione, ho deciso di implementare l’approccio di spiegabilità *SHAP* per evidenziare il contributo specifico di ogni feature (sintomo) nel determinare la classe (diagnosi di diabete). Ritengo tale scelta fondamentale per interpretare le decisioni del modello *Random Forest* che, pur garantendo performance d'eccellenza, è intrinsecamente complesso a causa della sua struttura a foresta decisionale.

L'utilizzo di dati non scalati per l'analisi *SHAP* è stato adottato per garantire che i risultati fossero direttamente interpretabili in termini clinici. Questo ha permesso di trasformare l'algoritmo in uno strumento di supporto decisionale trasparente, affidabile e coerente con la realtà medica, garantendo la piena leggibilità dei driver diagnostici.

```
# ------------------------------------------------------------------------------------------
# 5. Implementazione approccio di spiegabilità SHAP
# ------------------------------------------------------------------------------------------
plt.close('all') # pulizia ambiente grafico per ottenre una corretta rappresentazione del summary plot

X_test_df = pd.DataFrame(X_test, columns=X.columns) # utilizzo il dataframe NON scalato
explainer = shap.TreeExplainer(models["Random Forest"]) # creazione explainer su random forest
shap_values = explainer.shap_values(X_test_df) # calcolo SHAP values

if isinstance(shap_values, list): # verifico il tipo di risultato per capirre come estrare i valori legazi ai pazienti positivi
    val_to_plot = shap_values[1] # in caso di risultato tipo lista, ovvereo SHAP <= 0.40
else:
    val_to_plot = shap_values[:, :, 1] # in caso di risultato tipo array 3D, ovvero SHAP >= 0.41

# creazione Summary Plot
shap.summary_plot(
    val_to_plot,
    X_test_df,
    plot_type="dot",
    max_display=X_test_df.shape[1],
    show=False
)
# titolo
plt.title(
    "Analisi della spiegabilità dei Sintomi\n(SHAP Summary Plot – Classe Diabetico)",
    pad=20
)
# label ordinate
plt.xlabel(
    "Impatto sulla previsione del modello (valore SHAP)",
)
plt.tight_layout()
plt.savefig("shap_summary_plot.png", dpi=300, bbox_inches="tight")
plt.show()
```

Il *SHAP Summary Plot* evidenzia come i sintomi della polidipsia e della poliuria, posizionati in cima all'asse delle ordinate e caratterizzati da *valori SHAP* prevalentemente positivi (punti rossi concentrati sulla destra), siano quelli con il maggiore impatto sulla diagnosi di diabete. Altri sintomi che spingono il modello verso una diagnosi positiva, sebbene in maniera progressivamente meno determinante, sono la perdita di peso improvvisa, l'irritabilità, la paresi parziale e la polifagia

Un'analisi più attenta del grafico mostra inoltre che la presenza di alopecia e prurito tende invece a spingere il modello verso una diagnosi negativa. Infine, il sesso biologico, pur non influendo in modo alcuno per una diagnosi positiva, contribuisce ad una maggiore probabilità di diagnosi negativa per il sesso maschile.



Considerazioni finali

L’esercitazione ha permesso di sviluppare un sistema di supporto alla diagnosi del diabete attraverso un workflow completo di Machine Learning.

Partendo da un'analisi esplorativa e dalla riduzione della dimensionalità tramite PCA, si è passati all'addestramento di tre modelli: *Logistic Regression*, *SVM* e *Random Forest*.

Il *Random Forest* si è rivelato la scelta ottimale in quanto, grazie alla spiegabilità con SHAP, coniuga prestazioni eccellenti ad una chiara interpretabilità dei fattori di rischio, elemento indispensabile per l'accettazione di tali tecnologie in ambito medico.