How I Trained Your Model:

approccio ibrido di machine learning e ragionamento ontologico per la classificazione degli episodi di “How I Met Your Mother”

Gruppo di lavoro:

- Francesca Cicala, 776356, f.cicala9@studenti.uniba.it

Link progetto github: https://github.com/fraacicala/Progetto_ICON25-26

AA 2025-26

How I Trained Your Model

Introduzione

Il dominio di interesse di questo progetto è la valutazione degli episodi di una famosa serie TV andata in onda dal 2005 al 2014: How I Met Your Mother. Essa è stata scelta per diversi motivi: non solo per interesse personale ma anche perchè, vista la sua longevità, è ricca di elementi semantici e dati statistici. Infatti verrà mostrato come l'integrazione di queste due componenti produca un modello di classificazione degli episodi più robusto e accurato.

Sommario

L'obiettivo del progetto è mostrare come un sistema ibrido, che integra apprendimento supervisionato e ragionamento automatico, sia un approccio molto più efficiente che basarsi solo su dati statistici. Il progetto è sviluppato su due fronti:

1. La prima parte riguarda l'addestramento e la valutazione di diversi modelli di classificazione su un dataset composto da metadati tecnici.
2. La seconda parte riguarda il ragionamento automatico: viene sviluppata un'ontologia in OWL contenente elementi narrativi della serie. Successivamente si

utilizza un reasoner per dedurre la classificazione semantica di ogni episodio (Eccellente, Buono, Scarso).

Infine si mostra l'impatto di questa nuova conoscenza nel momento in cui viene integrata come nuova feature nel dataset di partenza.

Elenco argomenti di interesse

Il progetto si focalizza su analisi e integrazione di due moduli fondamentali dell'Ingegneria della Conoscenza:

- Apprendimento supervisionato: vengono addestrati due modelli su metadati tecnici, utilizzando tecniche robuste di selezione di iperparametri e valutandoli tramite diverse metriche e grafici. I risultati sono messi a confronto e viene commentata la performance in assenza di conoscenza di dominio.
- Rappresentazione della conoscenza e ragionamento automatico: rappresenta la soluzione al problema evidenziato nella sezione precedente. Viene costruita una base di conoscenza ontologica in OWL sfruttando il ragionamento automatico per derivare conoscenza di alto livello.

Il progetto culmina nel capitolo delle conclusioni, dove viene mostrata l'integrazione tra questi argomenti e il successo ottenuto, realizzando un sistema ibrido ML+OntoBK.

Apprendimento supervisionato

Sommario

L'obiettivo di questo modulo è predire il successo di un episodio di How I Met Your Mother, basandosi unicamente su dati statistici. Questi dati, che poi costituiranno le features, sono presi da un dataset strutturato. La variabile target è il rating dell'episodio che viene trasformato in un problema di classificazione con "eccellente", "buono", "scarso".

Strumenti utilizzati

Il progetto è stato implementato in python con l'utilizzo di librerie quali:

- `pandas`: utilizzata per la manipolazione del dataset in formato CSV;
- `matplotlib`: utilizzata per creare e visualizzare grafici;

- `numpy`: utilizzata per gestire array
- `scikit-learn`, che comprende:
  - `LabelEncoder`: utilizzata per trasformare feature target testuali in valori numerici;
  - `DecisionTreeClassifier` e `RandomForestClassifier`: gli algoritmi di classificazione scelti per l'addestramento;
  - `GridSearchCV`: utilizzata per l'ottimizzazione degli iperparametri;
  - `StratifiedKFold`: utilizzata come strategia di suddivisione dei dati;
  - `cross_validate`: utilizzata per eseguire il ciclo esterno della validazione incrociata;
  - `learning_curve`: usata nella funzione di visualizzazione della curva di apprendimento.

Decisioni di progetto

Si è scelto un dataset strutturato trovato su Kaggle, contenente feature quali:

- Season: numero di stagione;
- Episode: numero di episodio;
- DateAired: data di rilascio dell'episodio;
- Title: titolo dell'episodio;
- Director: regista dell'episodio;
- Writers: sceneggiatori;
- Viewers: numero di spettatori;
- Votes: numero di voti;
- IMDB_Rating: valutazione dell'episodio.

Per prima cosa ho controllato che i dati fossero corretti: consultando il sito ufficiale dell'IMDB mi sono resa conto che il punteggio dell'IMDB_Rating era circa dello 0,2 in più nel mio dataset, perciò ho corretto i dati. Successivamente ho effettuato un preprocessing del dataset: per l'addestramento dei modelli ho deciso di non tenere in considerazione le feature "DateAired" e "Title" in quanto non rilevanti per la classificazione dell'episodio. Inoltre, ho convertito le feature testuali, "Director" e Writers, in valori numerici tramite LabelEncoder. Ho selezionato come target la feature "Verdict", che è stata ottenuta discretizzando la feature "IMDB_Rating", inizialmente

composta da valori reali, in tre classi: “Eccellente”, “Buono”, “Scarso”. Ovviamente IMDB_Rating è stata esclusa dal set di training.

I modelli di apprendimento supervisionato prendono le feature di input, le feature target e i dati di addestramento e restituiscono predittori, ovvero funzioni sulle feature di input che prevedono i valori per le feature target. A seconda della natura della feature target, l'apprendimento supervisionato si articola principalmente in due compiti:

1. Regressione: quando la feature target è una variabile continua;
2. Classificazione: Quando la feature target è una variabile categorica.

Per questo progetto, l'obiettivo è predire una categoria discreta, pertanto è stato adottato un approccio di classificazione. Ho selezionato e confrontato due modelli basati su alberi decisionali: il Decision Tree Classifier e il Random Forest Classifier.

- Il decision tree è un classificatore semplice le cui foglie rappresentano le classi di appartenenza mentre i nodi rappresentano le condizioni. È basato su regole if then else ed è utile a stabilire un buon punto di partenza, ma tende ad alta varianza e overfitting.
- Il random forest invece è un modello più complesso che ho scelto per cercare di superare i limiti del decision tree. Esso infatti consta di una serie di alberi di decisione che vengono addestrati e le cui previsioni poi vengono combinate secondo un criterio di ottimalità.

Per entrambi i modelli ho condotto un’ottimizzazione degli iperparametri tramite la definizione di una griglia. In particolare, per il decision tree:

```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5, 6],
    'min_samples_split': [5, 10, 15, 20, 25],
    'min_samples_leaf': [2, 3, 5, 7, 10, 15]
}
```

- `criterion`: misura la qualità di uno split. Sono stati testati sia ‘gini’ che ‘entropy’ per completezza.

- `max_depth`: rappresenta la profondità massima dell'albero. Ho scelto appositamente valori bassi al fine di mantenere l'albero semplice per evitare il forte rischio di overfitting visto il dataset modesto.
- `min_samples_split`: il numero minimo di campioni richiesti per dividere un nodo interno. Se fosse troppo basso, permetterebbe di creare nuovi split basati su pochissimi campioni. È stato impostato in modo che le regole vengano create su gruppi sufficientemente grandi.
- `min_samples_leaf`: il numero di campioni richiesti per essere un nodo foglia. Non ho considerato l'1 ma numeri più alti affinchè ogni previsione sia basata su un gruppo di dati e non su un singolo, per la creazione di regole più robuste e generalizzabili.

Tutte queste scelte sono state fatte per evitare che l'albero diventi troppo grande e quindi finisca per imparare a memoria i dati.

Per il random forest invece:

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}
```

Ho utilizzato gli stessi parametri con l'aggiunta di:

- `n_estimators`: numero di alberi della foresta. Ho scelto un range alto per garantire la stabilità del modello.

Per i restanti, ho aumentato `max_depth` includendo profondità moderate e anche 'None' per cercare un giusto equilibrio, in quanto il random forest dovrebbe essere in grado di gestire meglio l'overfitting rispetto al decision tree. Invece, `min_samples_split` e `min_samples_leaf` sono stati tenuti più bassi per sfruttare la forza dell'ensemble.

Per ottenere una stima realistica e non ottimistica, è stata utilizzata una Nested Cross-Validation, che si svolge in:

- ciclo esterno: viene utilizzato `cross_validate` per suddividere l'intero dataset in 5 fold, di cui ad ogni interazione uno viene tenuto da parte come set di test finale mentre i restanti vengono usati come test di training.

- ciclo interno: viene utilizzata `GridSearchCV` sul set di training fornito dal ciclo esterno che testa in modo esaustivo tutte le combinazioni di iperparametri definite dalla `param_grid` tramite una validazione incrociata a 5 fold per identificare la configurazione ottimale.

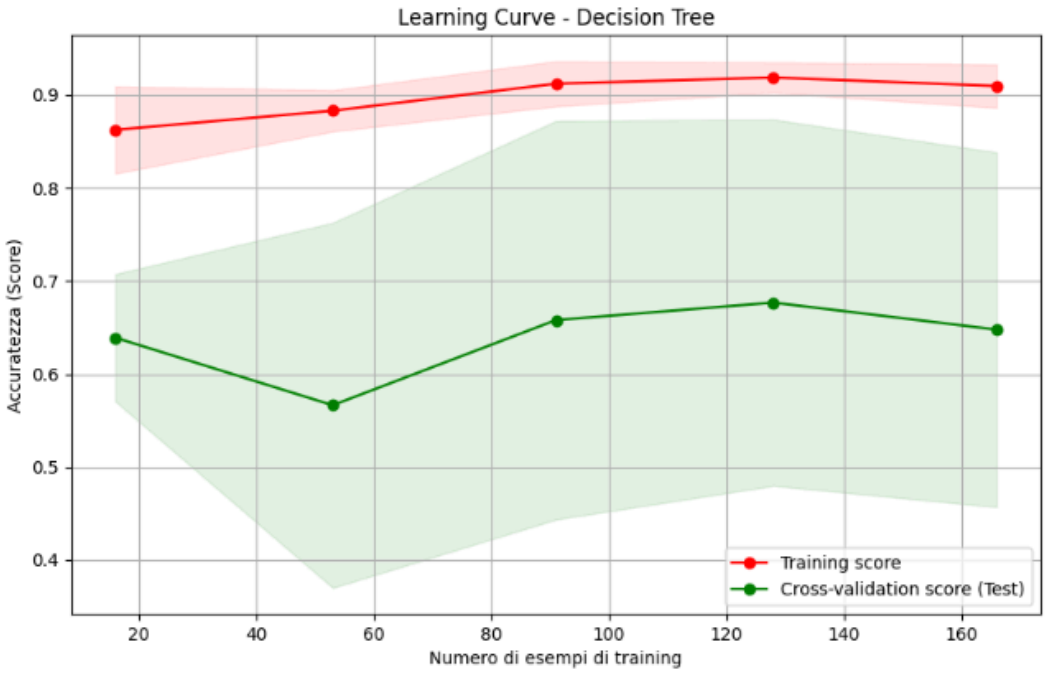
Alla fine del processo si ottiene una media dei punteggi ottenuti sui 5 set di test del ciclo esterno, modalità che fornisce una stima realistica. Per la valutazione dei modelli ho utilizzato le metriche accuracy, precision media, recall medio e F1-score medio. Inoltre, ho generato, grazie alla libreria `matplotlib`, grafici raffiguranti l'importanza delle feature e la curva di apprendimento. Tutti questi risultati saranno commentati nel seguente paragrafo.

Valutazione

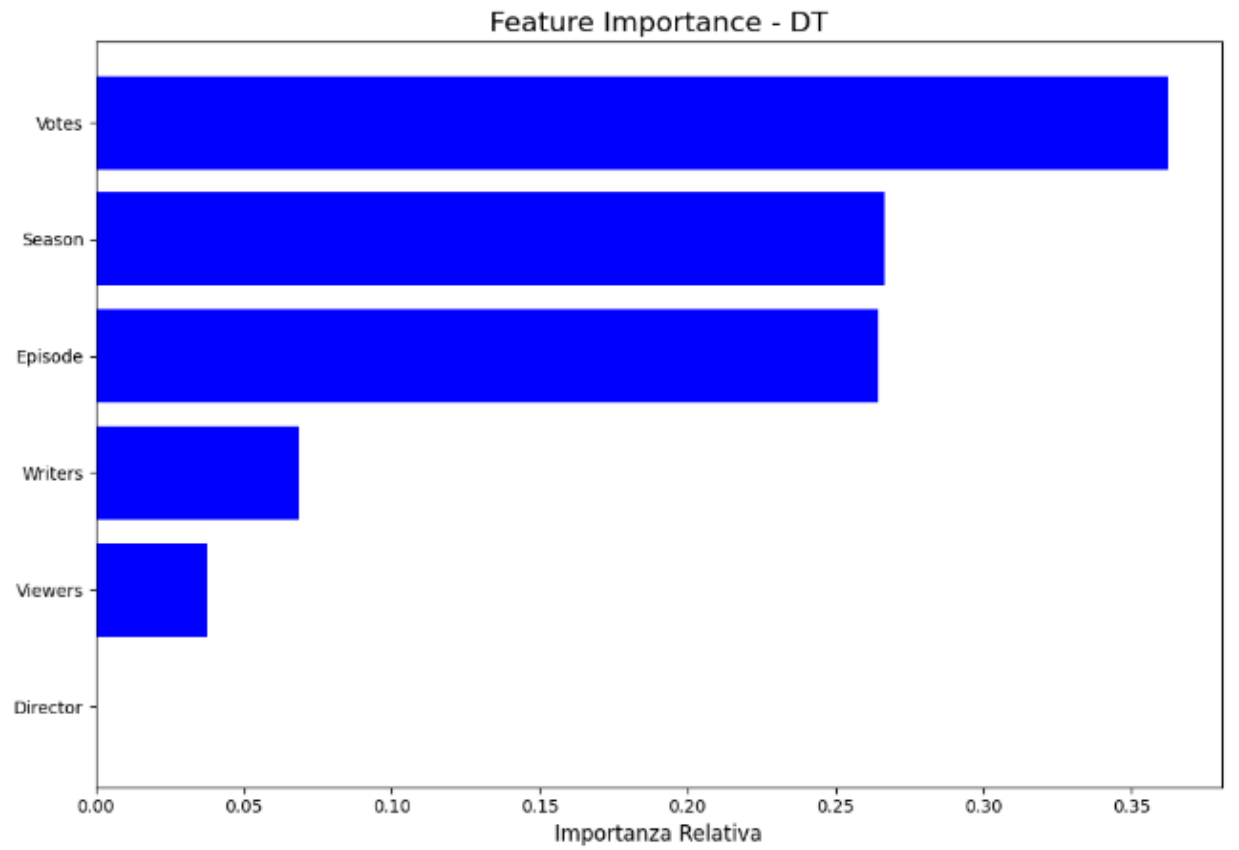
Per quanto riguarda il decision tree, le performance ottenute sono le seguenti:

```
Accuracy:  74.08% (+/- 7.04%)
Precision Media: 71.90%
Recall Medio:    74.08%
F1-Score Medio:  72.34%
Parametri migliori:  {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 3, 'min_samples_split': 5}
```

Notiamo che il decision tree sul dataset non ha delle metriche perfette: l'accuratezza del 74% indica che il modello riesce a classificare correttamente l'episodio circa 3 volte su 4. Tuttavia ha una deviazione standard del 7% che conferma l'instabilità del modello: le sue performance variano di molto a seconda della suddivisione dei dati. La precision è più bassa, il che significa che il modello non è estremamente affidabile. Il recall medio del 74.08% mostra un'efficace capacità di identificare i veri positivi, con una performance su questa classe in linea con l'accuratezza generale del modello. L'F1 score, media armonica tra precision e recall, conferma il bilanciamento del modello. Per la scelta dei parametri, il `GridSearchCV` ha selezionato una configurazione per combattere l'overfitting, soprattutto per quanto riguarda `min_samples_leaf` e `min_samples_split` in cui ha imposto che ogni foglia sia basata su almeno 3 campioni e che ogni split parta almeno da 5.



Il training score rappresenta l'accuratezza del modello quando viene testato sugli stessi dati su cui è stato addestrato. In questo caso essa è molto alta, quindi il modello riesce a memorizzare i pattern trovati. Il cross-validation score invece, rappresenta l'accuratezza del modello quando viene testato su dati mai visti prima. Nel grafico è bassa e instabile, segno che il modello fa fatica a generalizzare su dati nuovi.
L'elemento più denotabile è sicuramente l'enorme gap tra le due curve, il che significa che la conoscenza che il modello ha appreso dai dati di training non è trasferibile a nuovi dati, chiaro segno di overfitting.

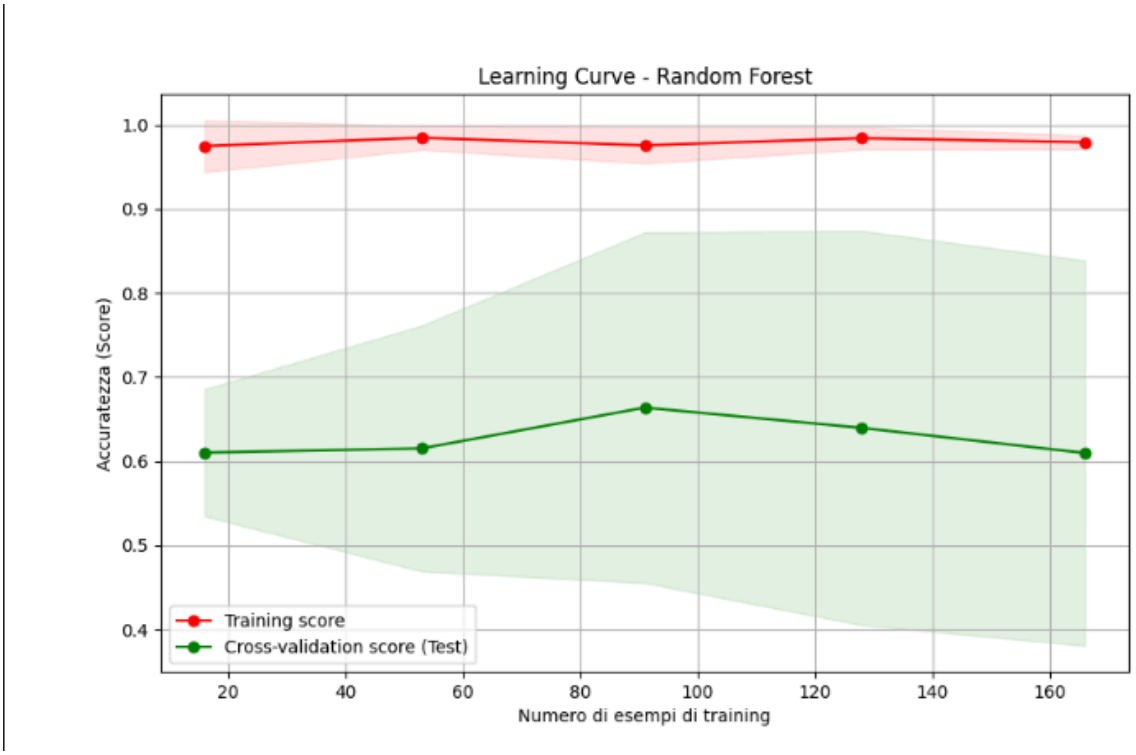


Notiamo come il decision tree abbia dato importanze diverse alle varie feature. Ha ritenuto che il numero di voti di un episodio fosse un dato importante per la predizione del successo: un episodio con tanti voti, e quindi molto discusso, sarà probabilmente significativo. Inoltre, presentano valori alti anche stagione ed episodio, quindi il voto di un episodio dipende anche dalla stagione in cui è contenuto e dalla sua posizione all'interno della stagione. Invece "Writers", "Viewers" e "Director" hanno valori più bassi. Sebbene il modello abbia identificato alcuni pattern deboli legati a "Writers", ha attribuito un peso molto basso al numero di spettatori, probabilmente considerandolo un indicatore meno affidabile di "Votes". Infine, la feature "Director" è risultata quasi del tutto irrilevante.

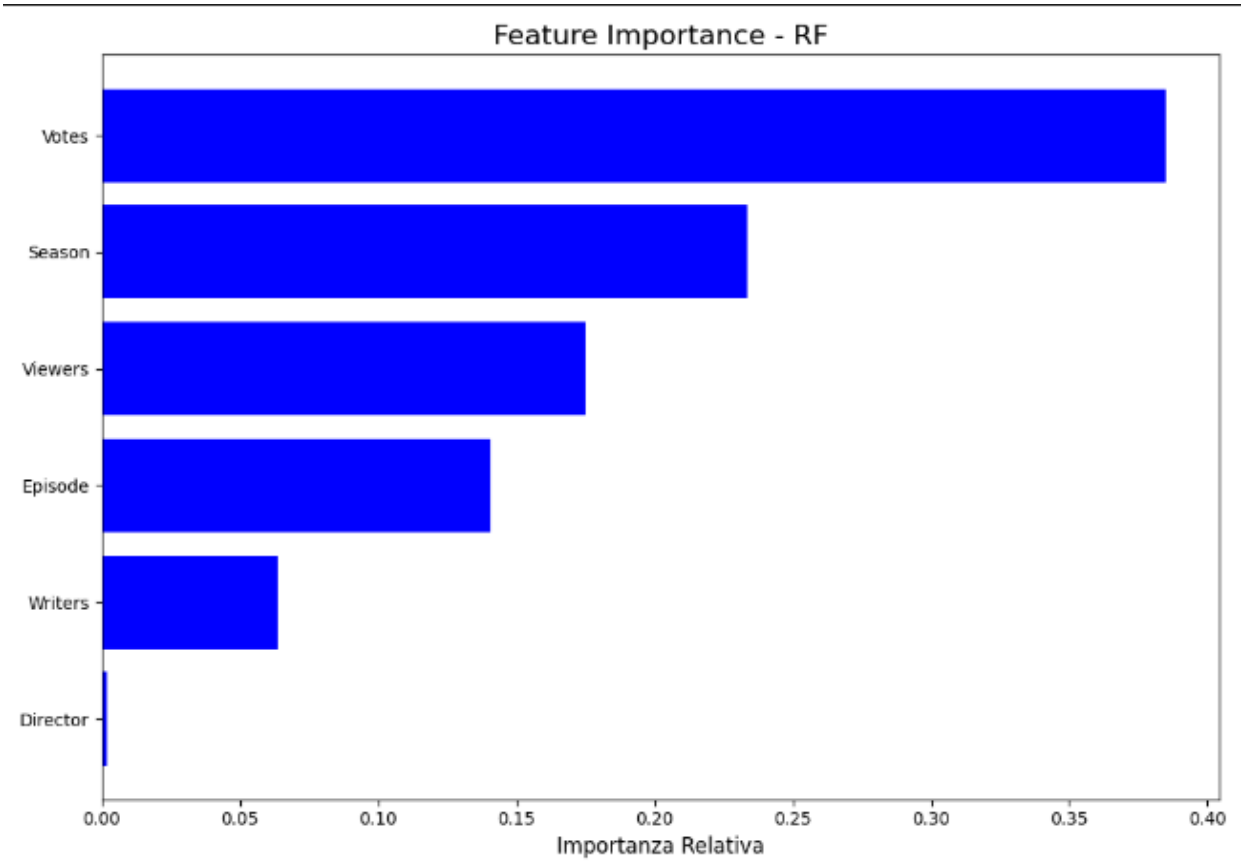
Per il random forest:

```
Accuracy:  75.52% (+/- 6.21%)
Precision Media: 75.94%
Recall Medio:    75.52%
F1-Score Medio:  74.98%
Parametri migliori:  {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50}
```

Il random forest rappresenta un leggero miglioramento rispetto al decision tree. Tutte le metriche riescono a raggiungere qualche punto in più, ma nonostante ciò la deviazione standard rimane alta, confermando l'instabilità. Per il resto, nonostante il modello sia più complesso, riscontra gli stessi problemi del decision tree. Si denota invece l'analisi degli iperparametri fatta da `GridSearchCV`: a differenza del decision tree che richiedeva una profondità limitata, il random forest permette a ogni albero di crescere fino alla sua massima profondità. Le restrizioni su `min_samples_ leaf` e `min_samples_split` sono più leggere, dovute alla natura stessa dell'ensemble che controlla meglio l'overfitting. Inoltre ha stimato che fossero necessari "solo" 50 alberi per arrivare a una stabilità.



Il punteggio di training è estremamente alto, quindi ogni albero memorizza la porzione di dati su cui viene allenato, per cui il punteggio su dati visti è sopra lo 0,9. Il punteggio di cross-validation mantiene un'accuratezza media, tra lo 0,6 e lo 0,7. L'ampio gap di generalizzazione e l'elevata varianza nei punteggi di cross-validation (indicata dalla larga banda di deviazione standard) suggeriscono che il modello è instabile e troppo complesso per la dimensione del dataset.



Il grafico conferma alcune feature del decision tree ma ne prende in considerazione altre: al primo posto c'è ancora il numero di voti, essenziale per capire la significatività di un episodio. Subito dopo ritroviamo la stagione, che dimostra il cambiamento di qualità che la serie ha avuto nel tempo. La novità principale è l'importanza acquisita da "Viewers": mentre il decision tree l'aveva quasi ignorata, con la random forest sale di grado. Infine, in fondo alla classifica, "Writers" e "Director" si confermano le variabili meno decisive per il modello.

Rappresentazione della conoscenza e ragionamento automatico

Sommario

Per la rappresentazione della conoscenza si è scelto l'utilizzo di un'ontologia in modo da rappresentare classi, proprietà e individui. Questo è stato fatto in linguaggio OWL, assicurando la presenza di una parte terminologica Tbox e una parte asserzionale Abox. L'obiettivo della mia base di conoscenza non è agire come un database: essa

rappresenta un sistema esperto in grado di classificare la qualità degli episodi della serie sfruttando il ragionamento automatico.

Strumenti utilizzati

Per la creazione e la gestione dell'ontologia è stato utilizzato l'editor Protégé. Una volta sviluppata in tutte le sue parti, è stata integrata in Python tramite la libreria `owlready2` , utilizzata per interagire con l'ontologia in formato RDF. Il motore di inferenza utilizzato è stato il reasoner Hermit.

Decisioni di progetto

La prima decisione importante a cui sono andata incontro è stata proprio la scelta di creare l'ontologia. Sebbene il web sia pieno di vaste basi di conoscenza come DBpedia o Wikidata, ho deciso di non utilizzarle per diverse ragioni: la più importante è stata la specificità del dominio. Avevo bisogno di un'ontologia che trattasse concetti propri dell'universo di How I Met Your Mother, non elementi generici di una qualsiasi serie TV, e questo sul web non era presente. Inoltre, la mia ontologia doveva comunque integrarsi con la parte di apprendimento supervisionato, quindi avevo bisogno di avere il pieno controllo sulla logica della Tbox per far sì che l'output del ragionamento fosse allineato con quello dell'apprendimento. Tuttavia, per garantire l'interoperabilità e mantenere un legame con il Web Semantico, ho utilizzato `rdfs:seeAlso` per collegare le mie classi più generiche a quelle effettive presenti su DBpedia. Per l'appunto la prima cosa che ho fatto è stata la definizione delle classi, che non sono state scelte casualmente ma dopo un'analisi del dominio: ho identificato un insieme di concetti che rendessero un episodio positivo o negativo, scelti non solo basandomi su un'opinione soggettiva ma anche su discussioni di altri utenti. Esse hanno tutte la stessa radice, ovvero `owl:Thing` , ma sono state sviluppate in sottoclassi differenti:

- *Elemento_narrativo*: contenente elementi di trama, sia positivi che negativi.
    - *Debolezza_narrativa*: raggruppa difetti che rendono un episodio poco apprezzato, come *Comportamento_tossico* dei personaggi, *Evento_fallito* che delude le aspettative degli utenti, *Trama_piatta* che caratterizza un episodio filler.
    - *Elemento_portante:* contiene elementi che rendono un episodio coinvolgente, come la presenza di *Arco_narrativo* (aneddoto che viene richiamato nel corso delle stagioni) o una *Teoria* (la serie è piena di teorie assurde ma divertenti).
- *Evento:* contiene avvenimenti concreti che avvengono all'interno di un episodio.

    - *Evento_trama*: eventi con un impatto diretto e significativo sulla trama come *Matrimonio, Notizia_spiacevole, Rottura_importante.*
    - *Struttura_temporale:* modi di raccontare l'episodio, come *Blackout* che crea mistero e curiosità e *Flashback_rivelatore,* tecnica utilizzata per fare rivelazioni importanti.
- *Oggetto_iconico*: la serie è piena di oggetti iconici che non possono essere trascurati, basti pensare che le due relazioni principali della serie si fondano su un *Corno_blu* e *Ombrello_giallo*.
- *Personaggio:* classificati in base alla loro importanza nella serie
    - *Personaggio_irrilevante*: personaggi secondari che non hanno avuto impatto sulla trama se non in modo negativo.
    - *Personaggio_rilevante:* può essere una *Guest_star* (personaggi interpretati da persone famose) e P*ersonaggio_trama,* personaggi fondamentali per la storia (esclusi ovviamente i protagonisti).
- *Episodio:* classe distinta in E*pisodio_eccellente, Episodio_buono, Episodio_scarso*. Queste classi non sono state popolate manualmente, ma rappresentano il target dell'inferenza logica del reasoner.

Ovviamente non sono riuscita a modellare ogni singolo dettaglio, è anche per questo che le classi hanno nomi specifici quali *Flashback_rivelatore* o *Rottura_importante*, create per distinguere eventi significativi da altri che rappresenterebbero solo rumore.

In seguito, ho popolato le classi: gli individui della classe Episodio (208 in totale) sono stati aggiunti tramite uno script in python; gli individui delle altre classi invece sono stati aggiunti manualmente, basandomi su fonti esterne affidabili come i wiki, per garantire oggettività.

Le relazioni sono state modellate tramite:

- Object properties, per collegare gli individui fra loro;
- Data properties, per associare dati letterali agli individui.

Ciò che permette al reasoner di inferire quali episodi siano eccellenti, buoni o scarsi è la definizione di regole di inferenza tramite il costrutto `Equivalent to`, in modo da definire le condizioni necessarie e sufficienti per l'appartenenza di un individuo a una classe. In particolare, un episodio è definito eccellente se contiene almeno un elemento considerato positivo. La regola che ho stabilito è:

```
Episodio
 and ((haElementoNarrativo some Elemento_portante) or
 (haEvento some Evento_trama) or
 (haEvento some Struttura_temporale) or
 (haOggetto some Oggetto_iconico) or
 (haPersonaggio some Personaggio_rilevante))
```

Lo stesso ragionamento è stato utilizzato per la classe *Episodio_scarso*, definito tale se contiene almeno un elemento considerato negativo:

```
Episodio
 and ((haElementoNarrativo some Debolezza_narrativa) or
 (haPersonaggio some Personaggio_irrilevante))
```

Invece, gli episodi buoni sono stati definiti per esclusione. Non è stato possibile definire questa classe tramite negazione logica (not) direttamente in OWL a causa della sua Open World Assumption: il reasoner non può concludere che un episodio non sia “Eccellente” solo perché non ha l'informazione esplicita. La classificazione degli episodi buoni è stata quindi gestita a livello applicativo in Python. A queste classi è stato applicato il disjoint, in quanto non è possibile che un episodio sia allo stesso tempo eccellente, buono o scarso. Stessa cosa è stata fatta con *Matrimonio* e *Rottura_importante*.

Per quanto riguarda Python, ho lavorato con la libreria `owlready2` che mi ha permesso l’interazione con classi, individui e proprietà, e di conseguenza l’avvio del processo di ragionamento con il motore Hermit, richiamato con una funzione specifica della libreria. Ho creato la funzione `populate_ontology()` per caricare la mia ontologia vuota, composta solo dalla struttura, e creare per ogni riga del dataset un individuo di tipo Episodio con un ID univoco. Ad ogni episodio erano già associate le sue data properties, come ad esempio il titolo. La fase di inferenza è poi stata svolta dalla funzione `run_reasoning()`, che ha il compito di caricare l’ontologia completa e far partire Hermit tramite la chiamata a `sync_reasoner()`. Una volta ottenuti i risultati, essi vengono mappati su un dizionario che poi viene utilizzato nel main.

Valutazione

La KB ontologica non è stata valutata come un classificatore a sé stante, ma per il suo valore all'interno del sistema ibrido. Il suo impatto è stato valutato all’interno del

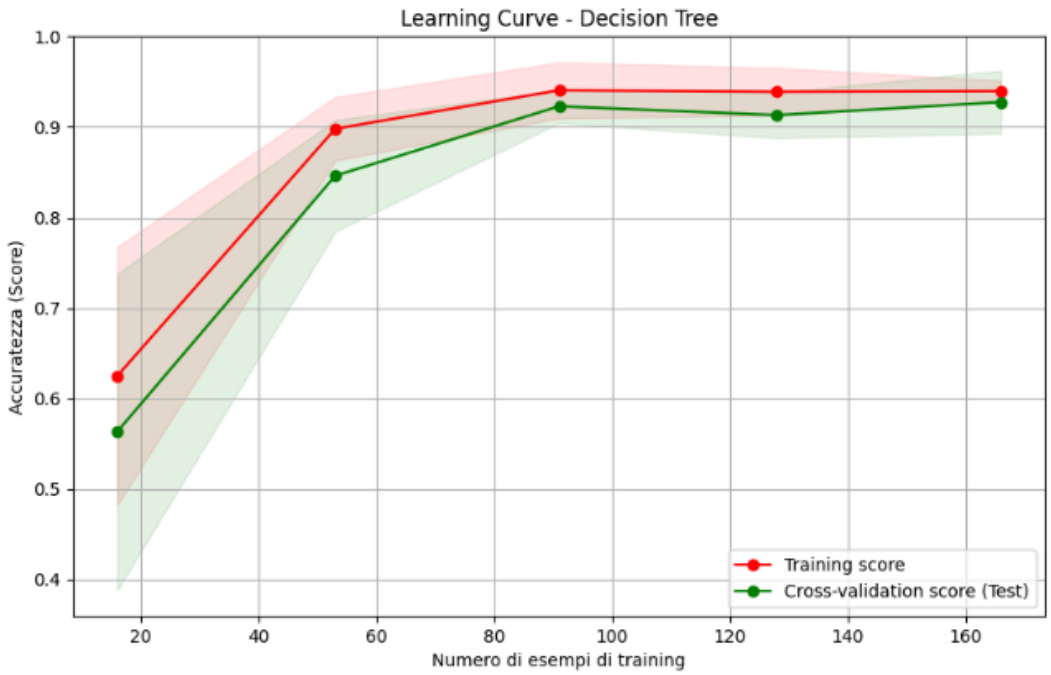
paragrafo conclusioni in cui viene dimostrato come l'output di questo sistema di ragionamento, utilizzato come feature ingegnerizzata, porti a un drastico miglioramento delle performance dei modelli di apprendimento.

Conclusioni

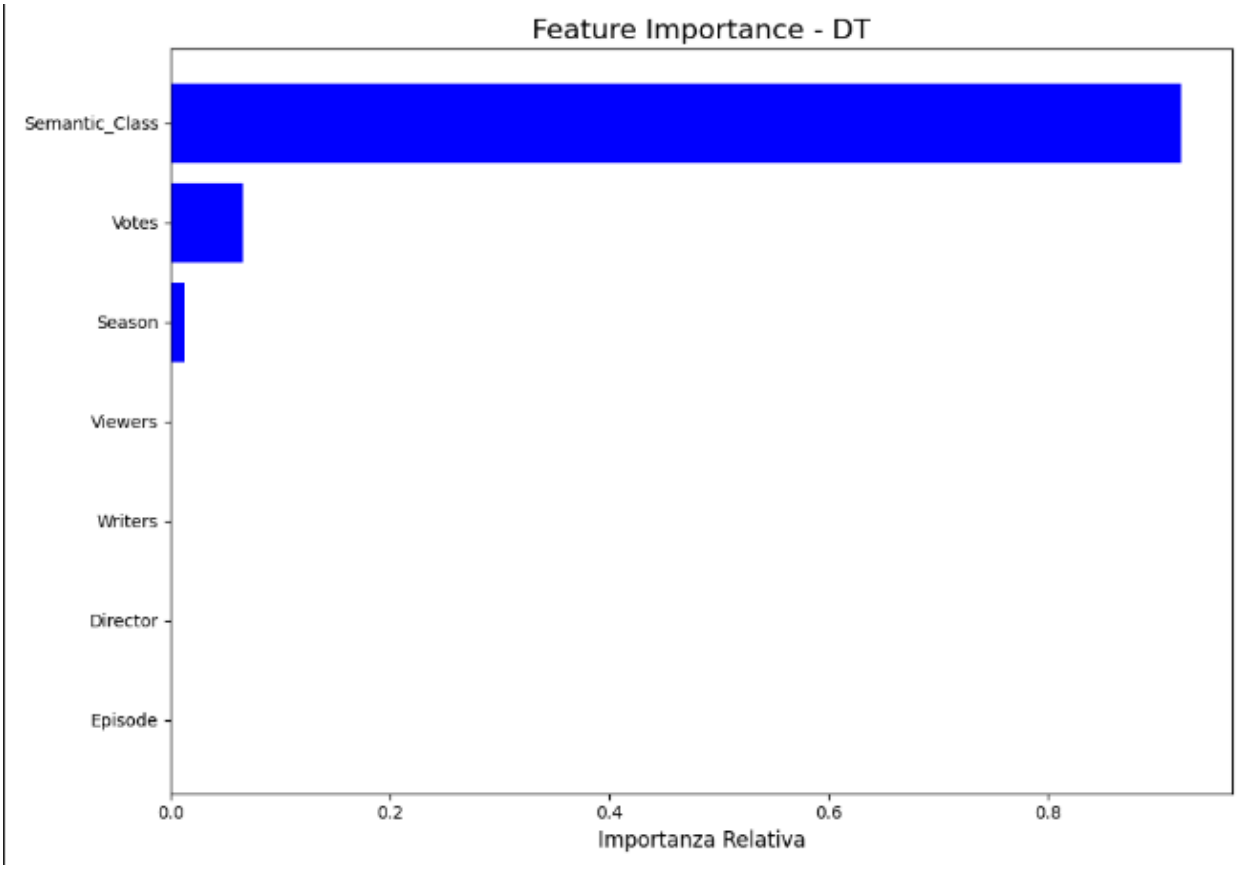
Il main del progetto è la parte che orchestra l'intero studio: richiama le funzioni che permettono il caricamento e il pre-processing del dataset e l'addestramento del Decision Tree e Random Forest, con la stampa delle relative metriche e grafici. Successivamente, richiama le funzioni per il caricamento e popolamento dell'ontologia e l'esecuzione del reasoner Hermit. Crea una nuova feature "Semantic_class" che viene aggiunta al dataset iniziale e infine riesegue l'addestramento dei modelli su questo nuovo dataset arricchito da conoscenza semantica. I risultati di questo ulteriore addestramento mostrano notevoli cambiamenti. Per il decision tree:

```
Accuracy:  92.79% (+/- 3.03%)
Precision Media: 93.63%
Recall Medio:    92.79%
F1-Score Medio:  92.82%
Parametri migliori:  {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 3, 'min_samples_split': 25}
```

I risultati denotano un miglioramento radicale e un successo dell'approccio ibrido: tutte le metriche hanno subito un notevole salto di qualità, di quasi 20 punti percentuali. Non solo, la deviazione standard è praticamente dimezzata rendendo il modello molto stabile oltre che accurato. Analizzando i parametri, si nota che la `max_depth` passa da 6 a 3, rendendolo più semplice, probabilmente perchè ritiene la nuova feature molto potente. L'iperparametro `min_samples_split` è passato da 5 a 25, quindi il modello ora crea nuove regole solo se basate su un gruppo consistente.



Il confronto con la curva di apprendimento del modello base è radicale. C'è una quasi perfetta convergenza tra le due curve, infatti il divario è stato annullato: ora la conoscenza appresa sui dati di training è trasferibile su nuovi dati. Entrambe arrivano ad un'accuratezza nettamente superiore, sopra il 90%, il che significa che l'aggiunta della nuova feature ha fornito un grande potere predittivo.

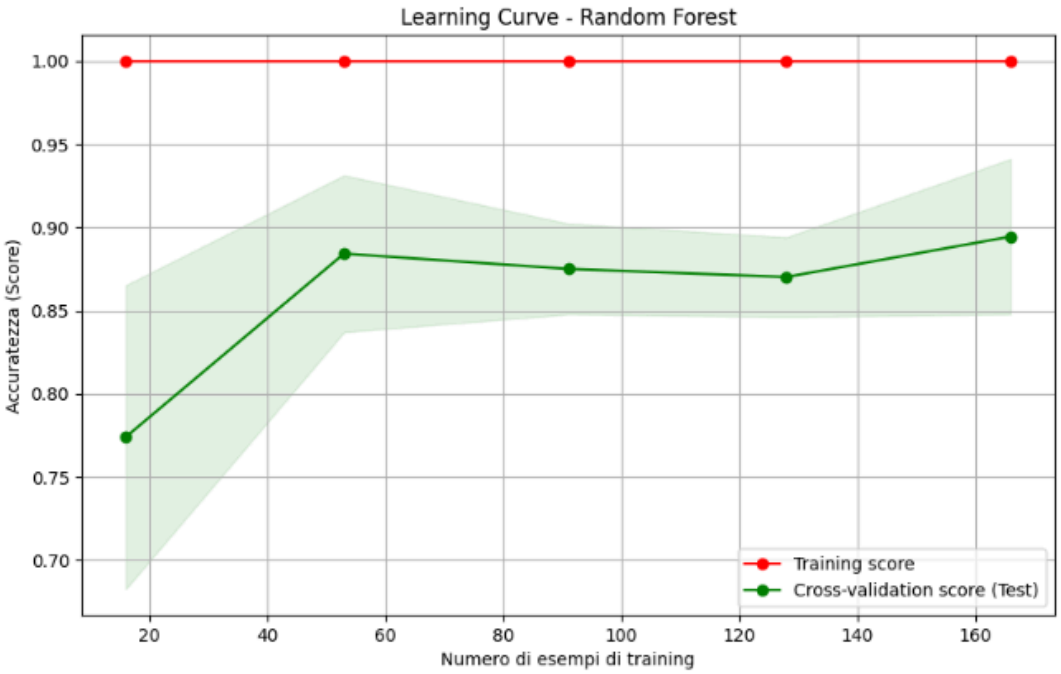


La feature “Semantic_Class” ha dominato l’intero scenario, assumendo un’importanza così elevata da rendere le altre quasi trascurabili.

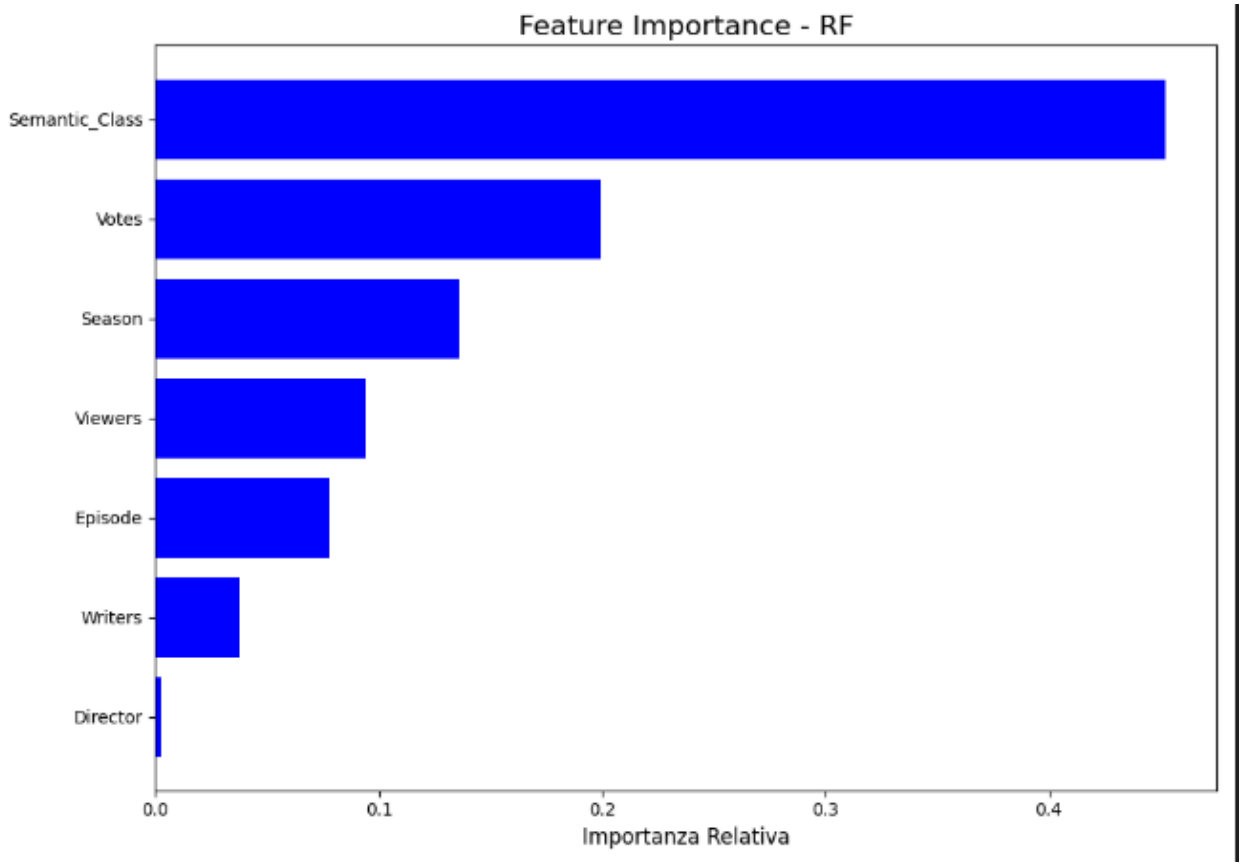
Per il random forest:

```
Accuracy:  91.84% (+/- 3.56%)
Precision Media: 92.66%
Recall Medio:    91.84%
F1-Score Medio:  91.83%
Parametri migliori:  {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
```

Anche qui si ha un netto miglioramento di tutte le metriche e il dimezzamento della deviazione standard. La parte più interessante riguarda il cambiamento dei parametri rispetto allo scenario precedente: `GridSearchCV` utilizza i parametri `min_samples_leaf: 1` e `min_samples_split: 2`. Questi sono i valori minimi possibili e, di fatto, rimuovono quasi ogni restrizione sulla crescita dei singoli alberi. Tuttavia, notiamo anche che il numero ottimale di alberi è passato da 50 a 150, quindi è triplicato, suggerendo che il modello ha sfruttato un numero maggiore di alberi per stabilizzare il voto finale.



Qui notiamo comunque un cambiamento rispetto al grafico precedente, ma non drastico come quello del decision tree: non c'è una convergenza tra le due linee, tuttavia il gap si è ridotto notevolmente. Il training score è al 100%, indicando che ogni albero della foresta impara a memoria la sua porzione di dati. Il punteggio di cross validation è salito notevolmente il che conferma che l'arricchimento del dataset ha funzionato.



Anche per il random forest la feature più rilevante è “Semantic_Class”, tuttavia ha un peso diverso rispetto al decision tree. In questo caso, il modello la ritiene la più significativa, ma comunque continua a dare un’importanza non trascurabile alle altre. La gerarchia finale delle feature è un ibrido tra il modello base e quello arricchito dalla conoscenza ricavata dall’ontologia dimostrando che il random forest sia riuscito a creare un modello più forte e bilanciato.

In conclusione, lo studio ha dimostrato che per comprendere il successo degli episodi di una serie TV i dati statistici da soli non sono sufficienti. L’integrazione con la conoscenza semantica della trama ha migliorato le prestazioni dei modelli, rendendoli più completi. Tuttavia, è importante notare che neanche i modelli finali raggiungono un’accuratezza massima: questo suggerisce che persino un’ontologia dettagliata fatica a formalizzare fattori come la qualità della scrittura di un episodio o l’impatto emotivo che genera sugli spettatori. Questo apre la porta a sviluppi futuri: sicuramente l’ontologia potrebbe essere espansa, specializzando le classi create e aggiungendone altre. Inoltre, come evidenziato, si potrebbero integrare tecniche di Sentiment Analysis, per spiegare anomalie come il finale della serie: l’ontologia lo classifica come eccellente, essendo ricco di eventi importanti, tuttavia gli ultimi due episodi sono i più

odiati dal pubblico. Infine sarebbe curioso testare altri modelli, cercando di gestire meglio lo sbilanciamento delle classi e ridurre il fenomeno di overfitting.

Riferimenti bibliografici

[1] Pandas: Getting Started. https://pandas.pydata.org/docs/getting_started/index.html#getting-started

[2] Pandas: User Guide. https://pandas.pydata.org/docs/user_guide/index.html

[3] Stack Overflow: convert text columns into numbers in sklearn. https://stackoverflow.com/questions/34915813/convert-text-columns-into-numbers-in-sklearn

[4] Scikit-learn: LabelEncoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[5] Scikit-learn: Nested versus non-nested cross-validation. https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

[6] Matplotlib: Quick Start. https://matplotlib.org/stable/users/explain/quick_start.html

[7] Scikit-learn: DecisionTreeClassifier https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[8] Scikit-learn: RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[9] Supervised Machine Learning: D. Poole, A. Mackworth: Artificial Intelligence: Foundation of Computational Agents. [Ch.7]

[10] Knowledge Graph and Ontologies: D. Poole, A. Mackworth: Artificial Intelligence: Foundation of Computational Agents. [Ch.16]