



SISTEMA INTELLIGENTE PER LA CLASSIFICAZIONE DELLA QUALITÀ DEL VINO

Realizzato da:

Claudio Valenziano

Matr: 706467

GitHub Repository:

<https://github.com/claudiovalen/progetto-icon.git>

IDE UTILIZZATO:

- PYCHARM

LIBRERIE IMPORTATE:

- Numpy
- Pandas
- Matplotlib
- Seaborn
- Scikit-learn
- Pyke
- Pyswip
- Tkinter

INTRODUZIONE:

Il caso di studio mira ad aiutare le filiere produttive di vino nella determinazione della qualità del vino prodotto. In Italia, ad esempio, ci distinguiamo per essere i primi produttori al mondo di vino, ma non siamo ancora del tutto abituati all'utilizzo degli strumenti tecnologici a nostra disposizione. In particolare, questo sistema può

fungere da supporto sia alla figura dell'enologo nella valutazione della qualità del vino o nell'identificazione delle sue malattie, aiutandolo nell'effettuare controlli che richiedono tempo e meticolosità, ma anche al cliente stesso attraverso un sistema di ricerca di vini secondo i propri criteri. Per la valutazione del vino, attraverso un ampio set di dati contenente le sue proprietà chimiche, il sistema è stato addestrato a riconoscerne la buona o cattiva qualità.

DATASET:

Il dataset importato, presente nel repository GitHub, contiene le proprietà chimiche del vino che ne permettono la classificazione di qualità:

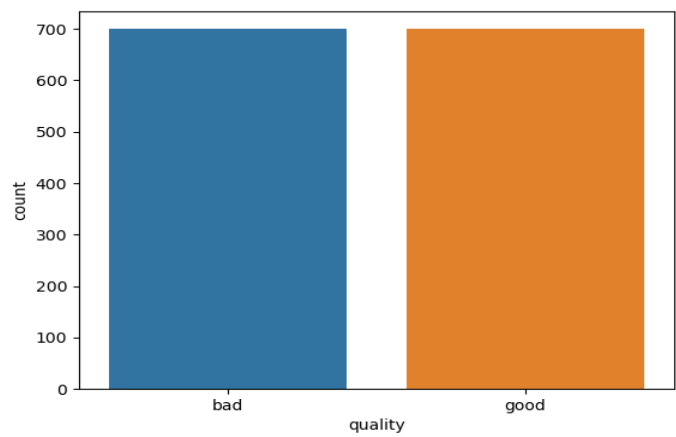
- 1 - acidità fissa: la maggior parte degli acidi coinvolti nel vino o fissa o non volatile (non evapora facilmente)
- 2 - acidità volatile: la quantità di acido acetico nel vino, che a livelli troppo alti può portare ad uno sgradevole sapore di aceto
- 3 - acido citrico: presente in piccole quantità, l'acido citrico può aggiungere 'freschezza' e sapore ai vini
- 4 - zucchero residuo: la quantità di zucchero che rimane dopo l'arresto della fermentazione (è raro trovare vini con meno di 1 grammo/litro e vini con più di 45 grammi/litro sono considerati dolci)
- 5 - cloruri: la quantità di sale nel vino
- 6 - anidride solforosa libera: la forma libera di SO₂ esiste in equilibrio tra SO₂ molecolare (come gas disciolto) e ione bisolfito; previene la crescita microbica e l'ossidazione del vino
- 7 - anidride solforosa totale: quantità di forme libere e legate di SO₂; a basse concentrazioni, l'SO₂ è per lo più non rilevabile nel vino, ma a concentrazioni di SO₂ libera superiori a 50 ppm, l'SO₂ diventa evidente nel naso e nel gusto del vino
- 8 - densità: la densità del vino è vicina a quella dell'acqua a seconda della percentuale di alcol e zucchero
- 9 - pH: descrive quanto è acido o basico un vino su una scala da 0 (molto acido) a 14 (molto basico); la maggior parte dei vini sono tra 3-4 sulla scala del pH
- 10 - solfati: additivo del vino che può contribuire ai livelli di anidride solforosa gassosa (SO₂), che agisce come antimicrobico e antiossidante

11 - alcol: la gradazione alcolica del vino

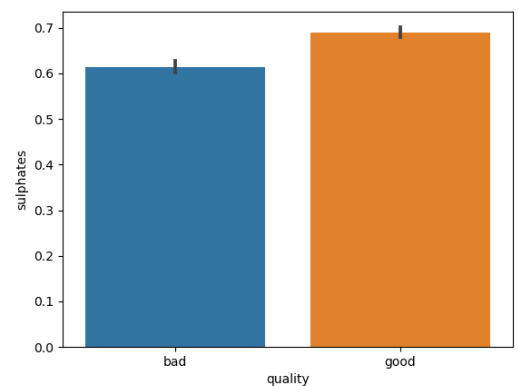
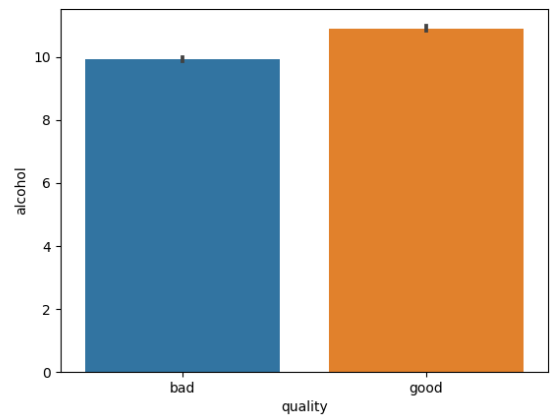
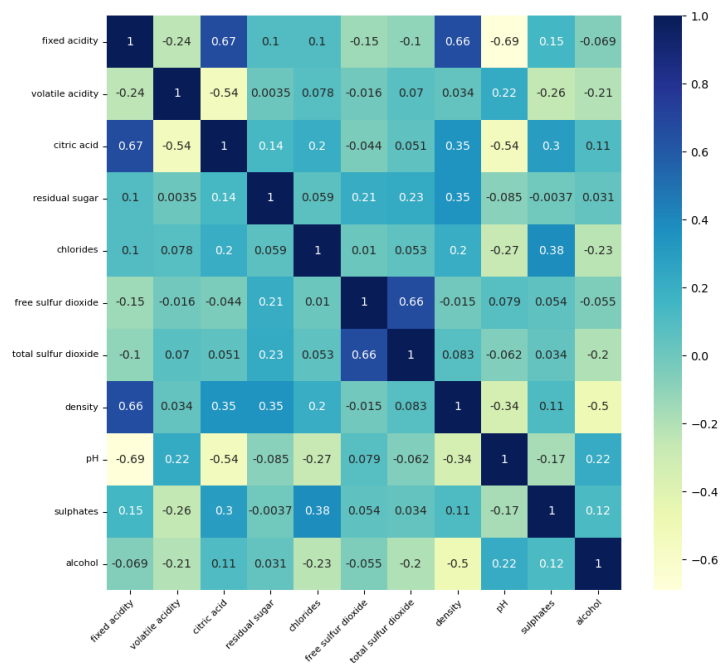
12 – quality: buona o cattiva qualità del vino

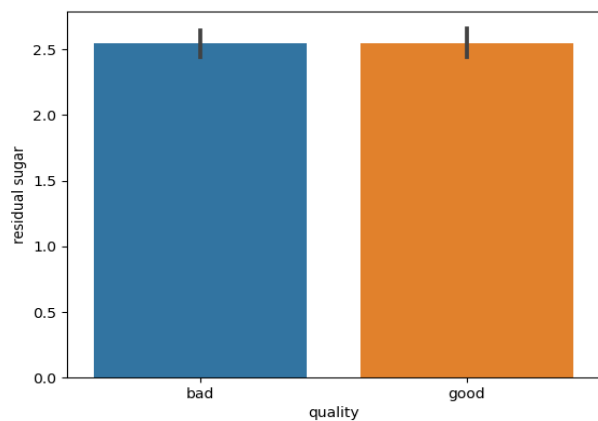
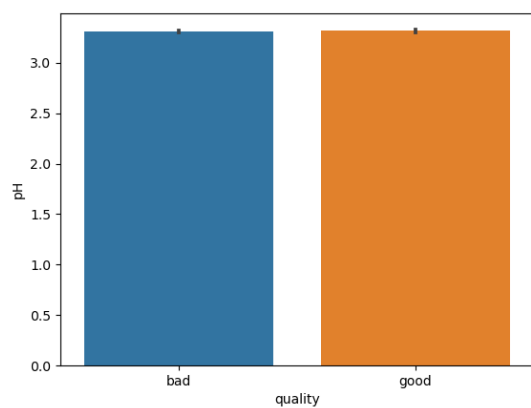
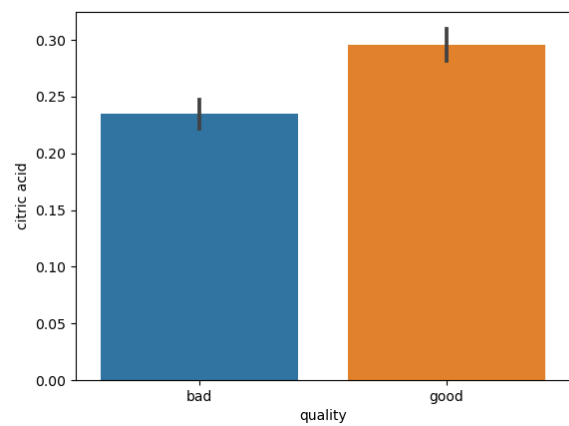
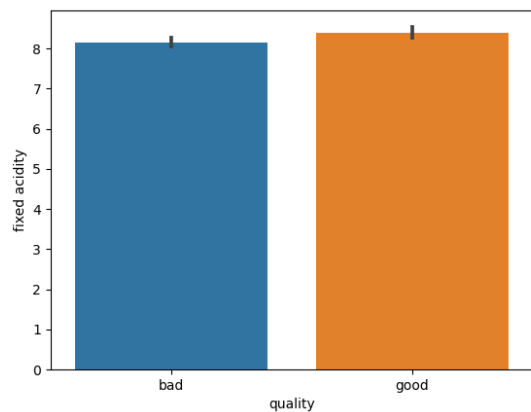
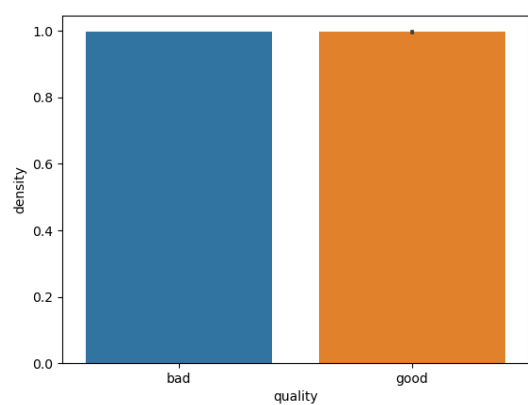
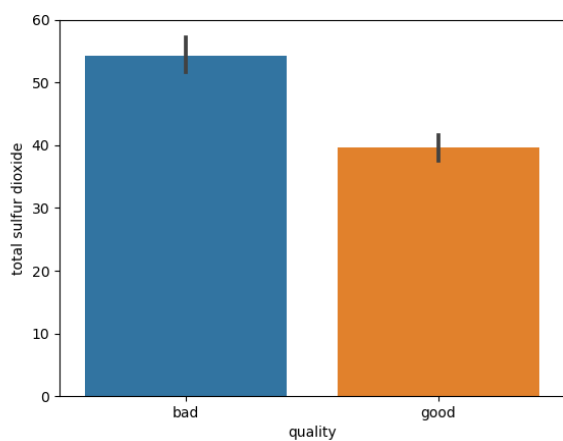
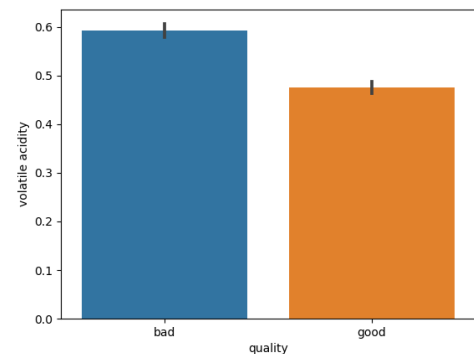
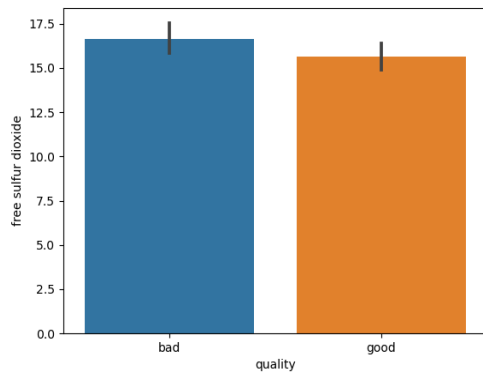
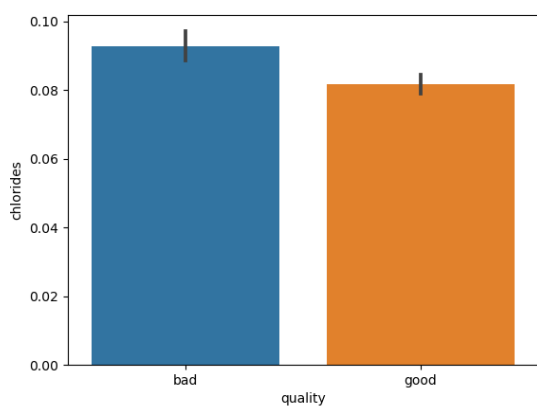
1	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
2	7.4	0.7	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	bad
3	7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.2	0.68	9.8	bad
4	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.997	3.26	0.65	9.8	bad
5	7.4	0.7	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	bad
6	7.4	0.66	0.0	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	bad
7	7.9	0.6	0.06	1.6	0.069	15.0	59.0	0.9964	3.3	0.46	9.4	bad
8	7.3	0.65	0.0	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	good
9	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	good
10	5.6	0.615	0.0	1.6	0.08900000000000001	16.0	59.0	0.9943	3.58	0.52	9.9	bad

Come si può vedere, la distribuzione delle classi di qualità all’interno del dataset è perfettamente bilanciata:



CORRELAZIONE DATI E INFERENZA:





```
plt.figure(figsize=(10,9))
map = sb.heatmap(df.corr(), annot=True, cmap="YlGnBu", xticklabels=True, yticklabels=True)
map.set_yticklabels(map.get_yticklabels(), rotation=0, fontsize=8)
map.set_xticklabels(map.get_xticklabels(), rotation=45, fontsize=8, rotation_mode='anchor', ha='right')
plt.show()

for i, col in enumerate(df.drop('quality', axis=1)):
    plt.figure(i)
    sb.barplot(x='quality', y=col, data=df)
    plt.show()
```

SCELTA FEATURES DI INTERESSE:

Emerge dunque che:

- Ci sono molti dati che non manifestano differenze nei valori a seconda della qualità, come ad esempio più di tutti il pH, lo zucchero residuo e la densità, che posso quindi non considerare nella classificazione, riducendo così il numero elevato di features.
- L'anidride solforosa totale, l'acidità volatile e l'acido citrico sembrano avere una correlazione più elevata con la qualità.
- Risulta che più alcool, maggior presenza di solfati e acido citrico comportano una qualità superiore, mentre maggior anidride solforosa totale e più acidità volatile portano ad una qualità peggiore.
- Non vi è una feature che ricopre un ruolo dominante nella determinazione della classe, anzi, i valori risultano in alcuni casi molto vicini.

CLASSIFICAZIONE QUALITÀ:

```
# Accuracy VS KNN
scores = []
error = []
for k in interval:
    knn = KNeighborsClassifier(p=k)
    knn.fit(X_train, Y_train)
    predictions = knn.predict(X_test)
    scores.append(accuracy_score(Y_test, predictions))
    error.append(np.mean(predictions != Y_test))

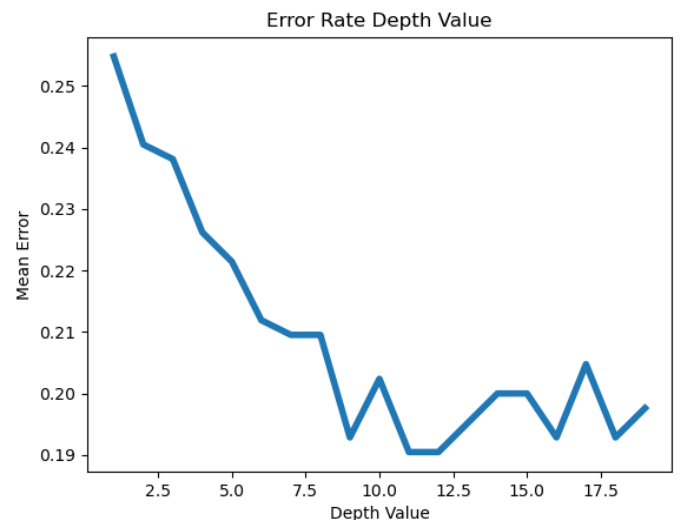
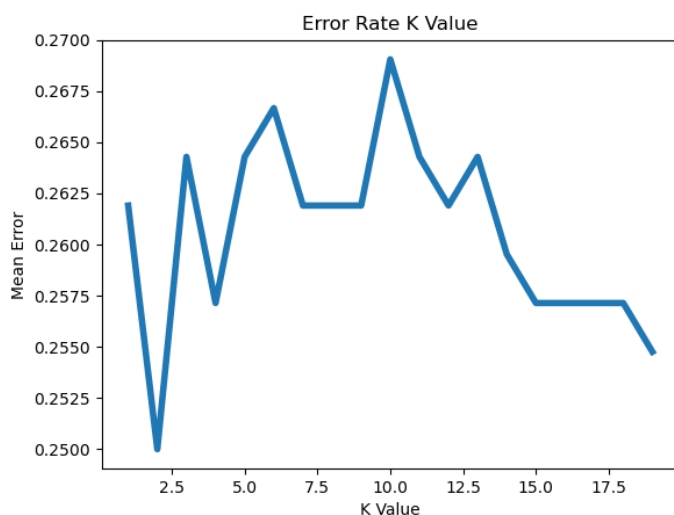
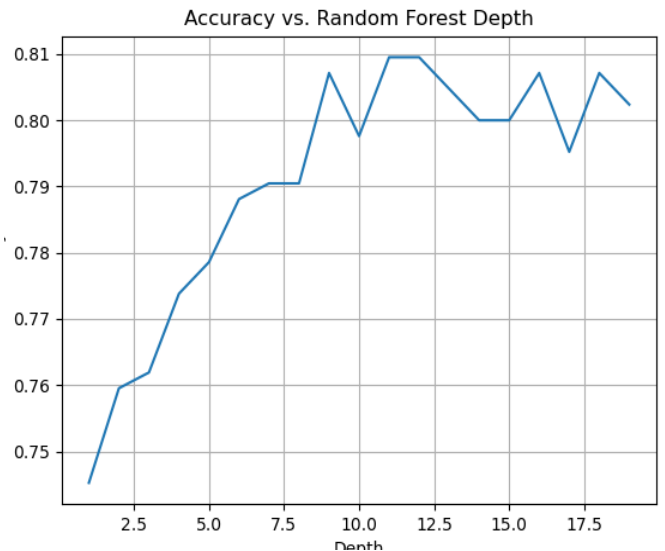
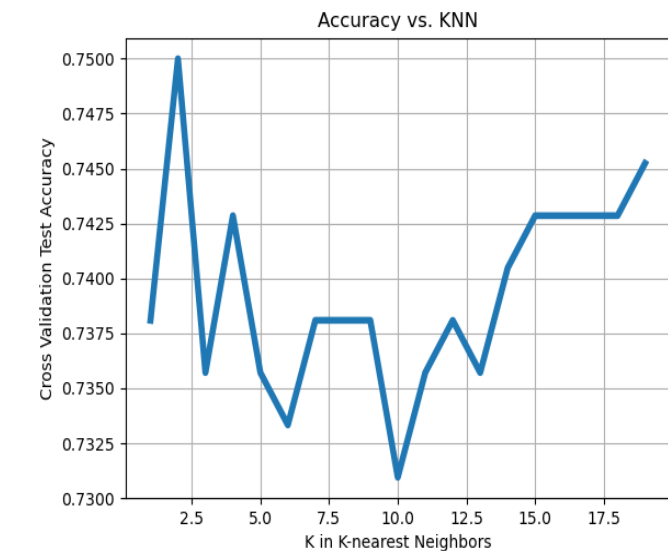
plt.plot(interval, scores, linewidth=4, markersize=10)
plt.grid()
plt.title('Accuracy vs. KNN')
plt.xlabel('K in K-nearest Neighbors')
plt.ylabel('Cross Validation Test Accuracy')
plt.show()

plt.plot(interval, error, linewidth=4, markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.show()
```

```
# Accuracy VS Depth Random forest Plot
accuracy = []
interval = np.arange(1, 20)
for i in interval:
    clf = RandomForestClassifier(max_depth=i, random_state=10)
    clf.fit(X_train, Y_train)
    predictions = clf.predict(X_test)
    accuracy.append(accuracy_score(Y_test, predictions))

plt.plot(interval, accuracy, linewidth=4, markersize=10)
plt.grid()
plt.title('Accuracy vs. Random Forest Depth')
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.show()
```

Per la classificazione della qualità del vino, dopo aver standardizzato le variabili numeriche ed aver rimosso le features superflue, ho utilizzato i seguenti algoritmi: il KNN e il Random Forest. Innanzitutto, ho verificato quale fosse la profondità ideale del Random Forest e il valore più appropriato per l'iperparametro K del KNN che garantissero così il risultato più accurato. Come si può notare dai risultati ottenuti, il Random Forest ha garantito una precisione più elevata:



Dall'analisi delle prestazioni degli algoritmi emerge che l'algoritmo Random Forest ottiene risultati migliori con una profondità compresa tra 11 e 12, mentre il KNN con un valore K tra 1 e 2. Utilizzando questi valori ottengo quindi le seguenti prestazioni e le seguenti matrici di confusione, che mi confermano la maggior efficienza dell'algoritmo Random Forest:

```
#Create a function within many Machine Learning Models
def models(X_train,Y_train,X_test,Y_test):

    #Using the Nearest Neighbor algorithm
    knn = KNeighborsClassifier(p=1)
    knn.fit(X_train, Y_train)
    knn_pred = knn.predict(X_test)
    knn_classification = classification_report(Y_test,knn_pred)

    #Using the Random Forest Classification algorithm
    forest = RandomForestClassifier(max_depth=11,random_state=10)
    forest.fit(X_train, Y_train)
    forest_pred = forest.predict(X_test)
    forest_classification = classification_report(Y_test,forest_pred)
```

```

model=models(X_train,Y_train,X_test,Y_test)
for i in range(len(model)):
    cm = plot_confusion_matrix(model[i],X_test, Y_test)
    plt.show()

```

```

[6]Random Forest Classifier Test Accuracy:
      precision    recall  f1-score   support

    bad         0.77      0.85      0.81       198
    good         0.85      0.77      0.81       222

 accuracy          0.81          0.81          0.81          420
 macro avg         0.81          0.81          0.81          420
weighted avg         0.81          0.81          0.81          420

```

```

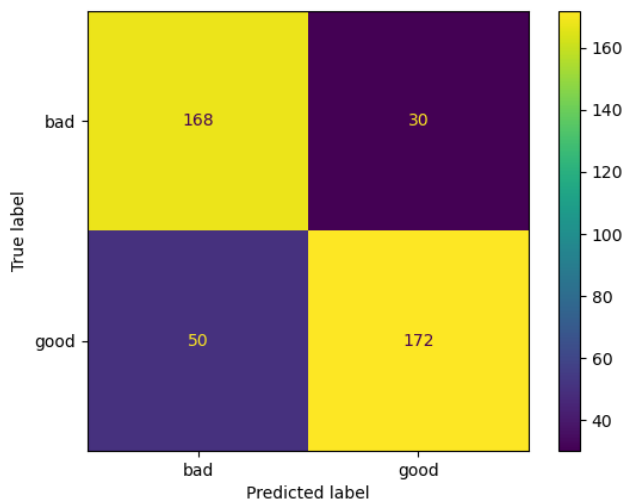
[1]K Nearest Neighbor Test Accuracy:
      precision    recall  f1-score   support

    bad         0.71      0.74      0.73       198
    good         0.76      0.73      0.75       222

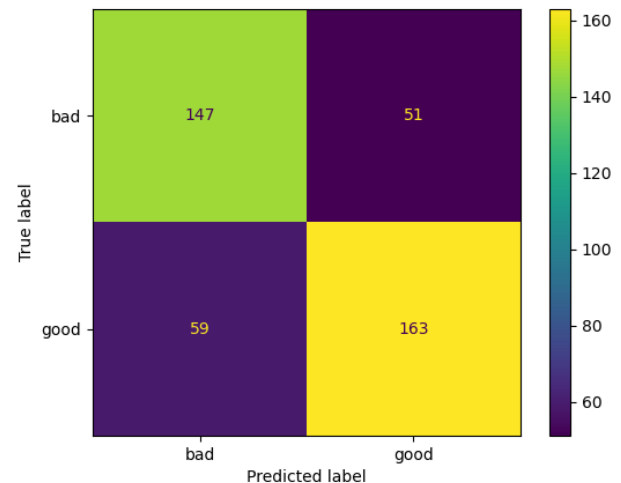
 accuracy          0.74          0.74          0.74          420
 macro avg         0.74          0.74          0.74          420
weighted avg         0.74          0.74          0.74          420

```

RANDOM FOREST:



KNN:



ONTOLOGIA:

Mi è sembrato utile realizzare una base di conoscenza da utilizzare per reperire i vini secondo certi parametri desiderati. Questa knowledge base è creata attraverso un algoritmo che legge i dati dei vini da un altro dataset da quello precedentemente utilizzato. Ho realizzato questo dataset raccogliendo informazioni su diversi vini appropriati a differenti contesti, ponendo maggior enfasi sui vini della nostra regione.

Ho realizzato un algoritmo che legge le caratteristiche dei vini dal dataset estraendo quelle di interesse.

Si è quindi implementata una knowledge base nella quale sono stati inseriti:

- nomi dei vini;
- tipi di vini (rosso, bianco, rosato, spumante);
- zona di produzione

- sapore (secco, fruttato, dolce, frizzante);
- budget
- cibo abbinabile

- Creazione KB:

```
# Generating nome - sapore
for row in dfKB.itertuples():
    nome = row[1]
    sapore = row[4]
    string = "nome-sapore(\"" + nome + "\",\"" + sapore + "\")."

    if is_empty(nome) and is_empty(sapore) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - prezzo
for row in dfKB.itertuples():
    nome = row[1]
    prezzo = row[5]
    string = "nome-prezzo(\"" + nome + "\",\"" + prezzo + "\")."

    if is_empty(nome) and is_empty(prezzo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - tipo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    string = "nome-tipo(\"" + nome + "\",\"" + tipo + "\")."
```

```
# Generating nome - tipo - prezzo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    prezzo = row[5]
    string = "nome-tipo-prezzo(\"" + nome + "\",\"" + tipo + "\",\"" + prezzo + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(prezzo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - tipo - cibo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    cibo = row[6]
    string = "nome-tipo-cibo(\"" + nome + "\",\"" + tipo + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"
```

```
# Generating nome - tipo - prezzo - cibo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    prezzo = row[5]
    cibo = row[6]
    string = "nome-tipo-prezzo-cibo(\"" + nome + "\",\"" + tipo + "\",\"" + prezzo + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(prezzo) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - sapore - prezzo
for row in dfKB.itertuples():
    nome = row[1]
    sapore = row[4]
    prezzo = row[5]
    string = "nome-sapore-prezzo(\"" + nome + "\",\"" + sapore + "\",\"" + prezzo + "\")."

    if is_empty(nome) and is_empty(sapore) and is_empty(prezzo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"
```

```
# Generating nome - cibo
for row in dfKB.itertuples():
    nome = row[1]
    cibo = row[6]
    string = "nome-cibo(\"" + nome + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - tipo - sapore
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    sapore = row[4]
    string = "nome-tipo-sapore(\"" + nome + "\",\"" + tipo + "\",\"" + sapore + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(sapore) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"
```

```
# Generating nome - tipo - sapore - prezzo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    sapore = row[4]
    prezzo = row[5]
    string = "nome-tipo-sapore-prezzo(\"" + nome + "\",\"" + tipo + "\",\"" + sapore + "\",\"" + prezzo + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(sapore) and is_empty(prezzo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - tipo - sapore - cibo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    sapore = row[4]
    cibo = row[6]
    string = "nome-tipo-sapore-cibo(\"" + nome + "\",\"" + tipo + "\",\"" + sapore + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(sapore) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"
```

```
# Generating nome - sapore - cibo
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    cibo = row[6]
    string = "nome-sapore-cibo(\"" + nome + "\",\"" + sapore + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(sapore) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating nome - sapore - prezzo - cibo
for row in dfKB.itertuples():
    nome = row[1]
    sapore = row[4]
    prezzo = row[5]
    cibo = row[6]
    string = "nome-sapore-prezzo-cibo(\"" + nome + "\",\"" + sapore + "\",\"" + prezzo + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(sapore) and is_empty(prezzo) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"
```



```
# Generating nome - prezzo - cibo
for row in dfKB.itertuples():
    nome = row[1]
    prezzo = row[5]
    cibo = row[6]
    string = "nome-prezzo-cibo(\"" + nome + "\",\"" + prezzo + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(prezzo) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"

file_data += "\n"

# Generating all
for row in dfKB.itertuples():
    nome = row[1]
    tipo = row[2]
    sapore = row[4]
    prezzo = row[5]
    cibo = row[6]
    string = "all(\"" + nome + "\",\"" + tipo + "\",\"" + sapore + "\",\"" + prezzo + "\",\"" + cibo + "\")."

    if is_empty(nome) and is_empty(tipo) and is_empty(sapore) and is_empty(prezzo) and is_empty(cibo) and (string not in file_data):
        file_data += string + "\n"
```

-Scrittura KB:

```
knowledge_base = open(KNOWLEDGE_BASE, mode="w")
knowledge_base.write(file_data)
knowledge_base.close()
print("\nFile created in: ", KNOWLEDGE_BASE)
```

-Funzioni di base della Knowledge Base

- Funzioni di gestione di clausole e query

Si sono create funzioni di aggiunta e cancellazione di clausole e di interrogazione della knowledge base in Prolog.

```
from pyswip import Prolog

prolog = Prolog()
prolog.consult("../dataset/knowledge.pl")
```

```
def addAssert(prolog, str):
    prolog.assertz(str)

def deleteAssert(prolog, str):
    prolog.retract(str)

def query(prolog, str):
    return list(prolog.query(str))
```

- Funzione di ricerca di vini in base a tipo/sapore/prezzo/abbinamento cibo:

```
def nomeTipoSearch(tip):
    tip = tip.lower()
    return query(prolog, "nome - tipo(X,\"\" + tip + "\"").")

def nomeSaporeSearch(sap):
    #a = input("Che sapore preferisci? [fruttato, secco]")
    sap = sap.lower()
    #print(query(prolog, "nome - sapore(X,\"\" + a + "\"")."))
    return query(prolog, "nome - sapore(X,\"\" + sap + "\"").")

def nomePrezzoSearch(prez):
    prez = float(prez)
    interval = np.arange(1, prez+1)
    my_list = []
    for i in interval:
        i = i.astype(str)
        if (query(prolog, "nome - prezzo(X,\"\" + i + "\"").") != []):
            my_list.append(query(prolog, "nome - prezzo(X,\"\" + i + "\"")."))
    return my_list

def nomeCiboSearch(abb):
    abb = abb.lower()
    return query(prolog, "nome - cibo(X,\"\" + abb + "\"").")
```

- Funzione di ricerca di vini in base a caratteristiche correlate:

Esempi:

```
def saporeCibo(sap,abb):
    sap = sap.lower()
    abb = abb.lower()
    return query(prolog, "nome - sapore - cibo(X,\"\" + sap + "\",\"\" + abb + "\"").")

def saporePrezzoCibo(sap,prez,abb):
    sap = sap.lower()
    abb = abb.lower()
    interval = np.arange(1, prez + 1)
    my_list = []
    for i in interval:
        i = i.astype(str)
        if (query(prolog, "nome - sapore - prezzo - cibo(X,\"\" + sap + "\",\"\" + i + "\",\"\" + abb + "\"").") != []):
            my_list.append(query(prolog, "nome - sapore - prezzo - cibo(X,\"\" + sap + "\",\"\" + i + "\",\"\" + abb + "\"")."))
    return my_list

def prezzoCibo(prez,abb):
    abb = abb.lower()
    interval = np.arange(1, prez + 1)
    my_list = []
    for i in interval:
        i = i.astype(str)
        if (query(prolog, "nome - prezzo - cibo(X,\"\" + abb + "\",\"\" + i + "\"").") != []):
            my_list.append(query(prolog, "nome - prezzo - cibo(X,\"\" + abb + "\",\"\" + i + "\"")."))
    return my_list
```

```
def allSearch(tip,sap,prez,abb):
    tip = tip.lower()
    sap = sap.lower()
    interval = np.arange(1, prez+1)
    abb = abb.lower()
    my_list = []
    for i in interval:
        i = i.astype(str)
        if (query(prolog, "all(X,\"" + tip + "\",\"" + sap + "\",\"" + i + "\",\"" + abb + "\").") != []):
            my_list.append(query(prolog, "all(X,\"" + tip + "\",\"" + sap + "\",\"" + i + "\",\"" + abb + "\")."))
    return my_list
```

INTERFACCIA GRAFICA

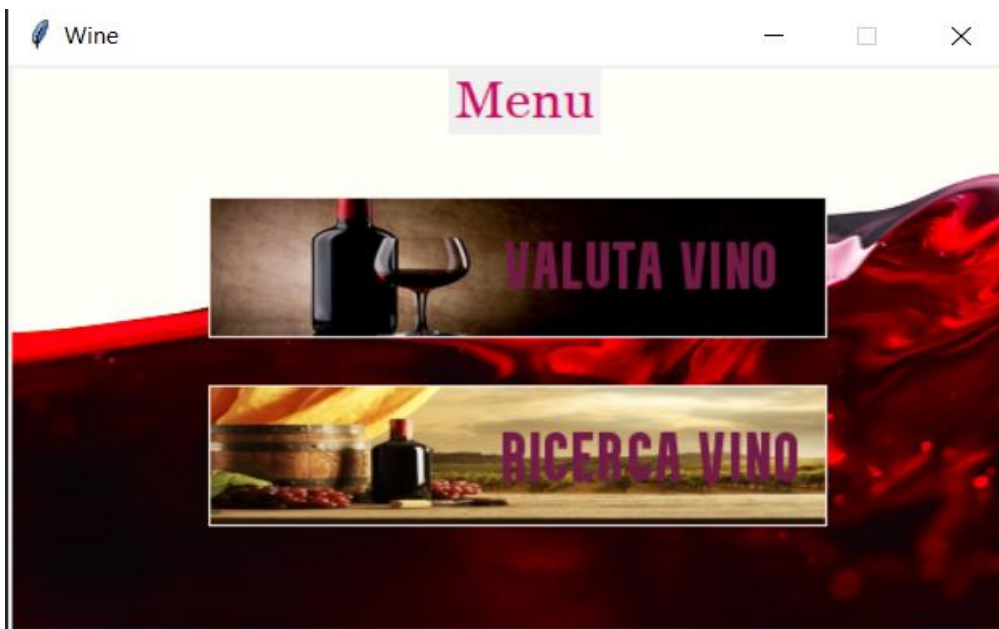
-Funzione e Realizzazione

Per consentire all'utente di interagire con le funzionalità del sistema, ho realizzato un'interfaccia dal facile utilizzo.

- Creazione finestra:

```
window = tk.Tk()
window.geometry("500x300")
window.title("Wine")
window.resizable(False, False)
window.configure(background="#525150")
```

- Realizzazione menu:



- **Valutazione qualità vino:**

Attraverso l'addestramento dell'algoritmo, ho realizzato una funzione che applichi l'algoritmo di predizione sui dati inseriti in input dall'utente restituendo in output la qualità predetta:



Ricerca vino

Inserimento Dati Vino

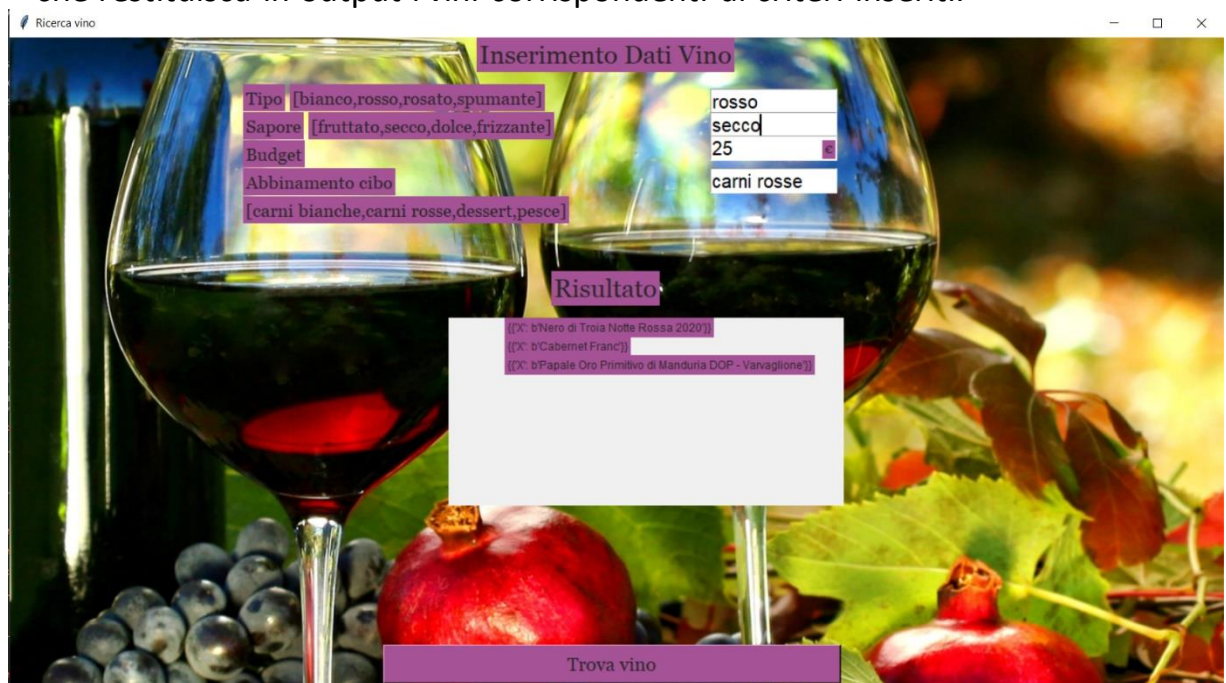
Fixed Acidity	6.7
Volatile Acidity	0.675
Citric Acid	0.07
Chlorides	0.089
Free Sulfur Dioxide	17
Total Sulfur Dioxide	82
Sulphates	0.54
Alcohol	10.1

['bad']

Qualità

- **Ricerca vino in base a parametri:**

Attraverso la consultazione della knowledge base, ho realizzato una funzione che restituisca in output i vini corrispondenti ai criteri inseriti.



Ricerca vino

Inserimento Dati Vino

Tipo [bianco,rosso,rosato,spumante]	rosso
Sapore [fruttato,secco,dolce,frizzante]	secco
Budget	25
Abbinamento cibo [carni bianche,carni rosse,desert,pesce]	carni rosse

Risultato

[[{"X": "bNero di Troia Notte Rossa 2020"}, {"X": "bCabernet Franc"}, {"X": "bPapale Oro Primitivo di Manduria DOP - Varaglione"}]]

Trova vino

KNOWLEDGE BASE PER IDENTIFICARE MALATTIE

Ho pensato poi che potesse essere utile per la figura dell'enologo la presenza di una funzione che sia d'ausilio al ritrovamento di malattie riguardanti la vite o il vino stesso. Ho quindi utilizzato una Knowledge Base frutto di un'attenta ricerca tramite il web circa i sintomi corrispondenti a determinate malattie.

Per progettare la costruzione e l'interrogazione di questa KB è stato utile, ad esempio, realizzare un albero decisionale.

Questa Knowledge Base viene creata con delle regole che si rifanno alla metodologia del Backward Chaining, affidandosi alle risposte dell'utente a determinate domande. Ciò è stato reso possibile tramite il sistema esperto Pyke. Sostanzialmente esso consente una programmazione logica in Python archiviando una base di conoscenza di fatti ed una base di domanda, applicando poi tramite un motore di inferenza le regole ai fatti.

```
def bc_test_questions():
    engine.reset() # Allows us to run tests multiple times.

    engine.activate('bc_simple_rules_questions')

    cont = 0
    try:
        with engine.prove_goal('bc_simple_rules_questions.quale_malattia($malattia)') as gen:
            for vars, plan in gen:
                print("Malattia: %s" % (vars['malattia']))
                cont = cont + 1

    except Exception:
        # This converts stack frames of generated python functions back to the
        # .krb file.
        krb_traceback.print_exc()
        sys.exit(1)

    if cont == 0:
        print("Malattia non riconosciuta")
```

I fatti, distinti tra specifici ed universali, vengono salvati su di un file di estensione .kfb.

Le regole invece vengono memorizzate su di un file di estensione .krb.

```
# bc_simple_rules.krb

what_malattia_Oidio
  use quale_malattia(Oidio)
  when
    questions.scelta_malattia($ans)
    check $ans in (1,)
    questions.clima_umido(True)
    questions.vedi_colore_grappoli(True)
    questions.colore_grappoli($ans2)
    check $ans2 in (2,)
    questions.acini_spaccati(True)

what_malattia_Flavescenza_Dorata
  use quale_malattia(Flavescenza_Dorata)
  when
    questions.scelta_malattia($ans)
    check $ans in (1,)
    questions.clima_umido(False)
    questions.vedi_colore_grappoli(True)
    questions.colore_grappoli($ans2)
    check $ans2 in (1,)
    questions.tralci_gommosi(True)
```

Infine le domande sono inserite su un file .kqb e, secondo le risposte dell'utente, viene richiamato il file .krb che determinerà la malattia corrispondente.

```
# questions.kqb

scelta_malattia($ans)
  La malattia riguarda l'uva o il vino
  ---
  $ans = select_1
  1: Uva
  2: Vino

clima_umido($ans)
  Il clima e' umido?
  ---
  $ans=yn
```

```
colore_grappoli($ans)
  Quale colore hanno i grappoli?
  ---
  $ans = select_1
  1: Dorato
  2: Grigiastro

acini_spaccati($ans)
  Gli acini si spaccano?
  ---
  $ans=yn
```


Dunque sulla base delle risposte fornite dall'utente alle domande viene stampata la malattia corrispondente:

```
La malattia riguarda l'uva o il vino?
```

```
1. Uva
```

```
2. Vino
```

```
? [1-2] 2
```

```
-----  
Il vino risulta opaco? (y/n) y
```

```
-----  
Noti sulla superficie un velo chiaro? (y/n) n
```

```
-----  
Il vino sembra torbido e dal colore smorto ? (y/n) y
```

```
-----  
Di che tipo e' il vino?
```

```
1. Rosso
```

```
2. Bianco
```

```
? [1-2] 2
```

```
Malattia: filante
```

CONCLUSIONE:

Si può pensare ad una futura implementazione del progetto anche in relazione alla nostra stessa regione Puglia, grande produttrice di vini in Italia, ma sicuramente meno al passo di altre regioni sul fronte tecnologico. Oltre ad un ausilio nella produzione e nella cura della vite e del vino, identificando quali attributi portano a valutazioni di qualità più alte o più basse, i risultati potrebbero essere utilizzati per aiutare i ristoranti a prevedere più facilmente quali vini apprezzeranno i clienti. I risultati delle ricerche dei vini basate su criteri potrebbero invece essere tenuti in conto per capire quali sono i parametri più richiesti dal cliente e capire quali categorie di vini importare maggiormente.

