Laurea Triennale in Informatica-Università di Salerno
Corso di *Fondamenti di Intelligenza Artificiale – Professori F. Narducci e F. Palomba*

WarehouseAI

Modulo di IA

Progetto combinato con IS: software NewDM

Sezione: Apprendimento

Autori: Cirillo Franco, Cirillo Luigi, Fusco Ciro, Aiello Vincenzo

Sommario

1. Introduzione

Ambito del modulo

Il nostro modulo è utilizzato nel prodotto software NewDM, una Piattaforma Desktop per la gestione di un punto vendita di una catena di supermercati. Il modulo è stato progettato per fornire un supporto all'attività del magazziniere, che tra le tante mansioni deve anche gestire le richieste di rifornimento per il supermercato.

Scopo del modulo

Il suddetto modulo di IA nasce allo scopo di fornire delle previsioni sulle quantità di prodotti realmente utili per il magazzino, al fine di dare dei consigli per l'acquisto di nuove forniture. In particolare, il magazziniere andrà ad inserire i dati richiesti e il sistema restituirà la quantità da acquistare per quel dato prodotto.

Specifica PEAS

Per una rappresentazione schematica dell'ambiente e dell'agente utilizziamo la specifica PEAS (*Performance, Environment, Actuators, Sensors*) che si basa sull'indicazione di quattro elementi:

- ***Performance*** (La misura di prestazione adottata per valutare l'operato dell'agente):
  buona previsione delle quantità di prodotti necessari al magazzino, riduzione dei costi e maggiori entrate per il magazzino.
- ***Environment*** (Descrizione degli elementi che formano l'ambiente):
  magazzinieri, rifornimenti.
- ***Actuators*** (Gli attuatori disponibili all'agente per intraprendere le azioni):
  mostra le quantità consigliate da rifornire.
- ***Sensors*** (I sensori attraverso i quali riceve gli input percettivi):
  input da tastiera

2. Analisi dei requisiti

Come prima fase abbiamo analizzato il comportamento che dovrebbe avere il nostro modulo, vogliamo fornire determinati input e vogliamo ottenere una previsione basata su di essi. Per fare ciò l'agente intelligente dovrebbe apprendere da un insieme di dati e quindi si dovrebbe addestrare su di essi per poter meglio reagire a nuovi dati.

L'insieme di dati (*dataset*) dovrebbe fornirgli sia input e sia l'output da dare, in modo da imparare al meglio possibile. Il nostro problema rientra precisamente nell'insieme dei problemi di Apprendimento supervisionato: apprendere una funzione che descrive il fenomeno studiato partendo da input e output noti. In sostanza si conoscono i dati e si conosce precisamente cosa essi rappresentano. Infatti, il nostro è principalmente un problema di *supply forecast* che sono soliti essere risolti con algoritmi supervisionati.

Lo step successivo, che è anche quello di maggiore interesse e su cui si basa l'intero modulo, consiste nel trovare le giuste: *features* (caratteristiche su cui l'agente andrà lavorare e che costituiscono l'input), *etichette* (risultato da dare in output) e quindi soprattutto trovare il dataset che servirà per l'addestramento.

Sono state effettuate numerose ricerche sul web al fine di trovare un dataset che potessimo utilizzare nel nostro modulo, ma non è stato trovato nulla di realmente adeguato al nostro scopo.

Si è optato per la costruzione di un dataset da zero, sebbene essendo a conoscenza che sarebbe stata un'attività abbastanza complessa che ci avrebbe occupato molto tempo, ma che ci avrebbe dato un buon risultato finale, perfettamente adattato al nostro scopo.

Attraverso un brainstorming abbiamo proposto numerose features e alla fine abbiamo selezionato le migliori, cioè quelle che a nostro avviso meglio caratterizzavano i dati. Invece per l'output abbiamo deciso di utilizzare un numero discreto di etichette, che rappresentano delle approssimazioni del quantitativo di rifornimento. Ulteriori spiegazioni verranno fornite nel paragrafo relativo al dataset.

A questo punto non ci resta che decidere il modello di apprendimento più adeguato al nostro utilizzo. Il modello in questione dovrebbe supportare l'apprendimento supervisionato e dalle nostre scelte (di output) dovrebbe essere un classificatore, quindi gestire output discreti.

Si è scelto cosi di utilizzare il classificatore Random Forest, ulteriori spiegazioni sono state fornite nel paragrafo relativo alle motivazioni.

Breve spiegazione Random Forest

Le Random Forest sono un metodo di ensamble learning per la classificazione e la regressione, che operano costruendo una moltitudine di alberi decisionali al momento dell'addestramento e fornendo la classe che è l'etichetta (classificazione) o previsione media (regressione) dei singoli alberi.

ALGORITMO

1. A partire dal training set D costruisci B sottoinsiemi di bootstrap (estrazione con reinserimento)
2. Per ciascuno dei B sottoinsiemi costruisci un albero di decisione
3. Ad ogni nodo
    - Seleziona m variabili in maniera random tra le M possibili e fallo in maniera indipendente per ciascun nodo
    - Trova il miglior criterio di split per tale sottoinsieme di m variabili
4. Fa crescere ciascun albero fino alla massima altezza in relazione al criterio di splitting adottato senza effettuare il pruning (taglio dei rami)

Quando un nuovo campione viene dato in input alla foresta, esso passa attraverso ciascun albero separatamente. La classe del nuovo campione verrà scelta per mezzo di un processo di voto o di media delle risposte di tutti gli alberi.

3. Soluzione proposta

Introduzione

La formulazione della soluzione proposta si può suddividere in 3 punti: dataset, notebook, integrazione con il software NewDM. È stato utilizzato github come strumento di versioning, tutto il software è raggiungibile al link: https://github.com/Ciro-Fusco/NewDM , il modulo al link: https://github.com/Ciro-Fusco/NewDM/tree/main/modulo. Sono state inoltre utilizzate tecniche di pair programming, cioè lavoro in coppia su un solo computer.

Dataset

Per effettuare la predizione si è scelto di utilizzare il classificatore Random Forest. In assenza di dati su cui effettuare l'addestramento di questo classificatore abbiamo provveduto alla scrittura delle features e di un dataset relativo, facendolo con la massima accuratezza in modo da renderlo il più realistico possibile.

Le features individuate dal team sono le seguenti:

- Tipologia (FruttaVerdura, Pesce, Carne, Casa, Elettronica)
- Stagione (Estate, Inverno, Primavera, Autunno)
- zona supermercato (Periferia, Residenziale)
- festività (Feriale, Lavorativo)
- scadenza (Breve, Media, Lunga)
- dimensione confezione (Piccola, Media, Grande)
- costo(prezzo)
- spedizione(prezzo)

Queste rappresentano tutte le variabili su cui deve basarsi il classificatore e che quindi devono essere immesse da input per ottenere una predizione. Informazioni aggiuntive:

- Tipologia: la categoria a cui appartiene il prodotto di cui vogliamo chiedere il rifornimento
- Stagione: stagione in cui chiediamo il rifornimento
- Zona supermercato: la zona in cui esso è situato
- Festività: se si richiede il rifornimento in un periodo feriale o lavorativo
- Scadenza: quella del prodotto
- Dimensione confezione: la dimensione della confezione del prodotto
- Costo: costo compresi i decimali
- Spedizione: costo per spedire il prodotto

L' etichetta che deve restituire il classificatore è fornitura (20-50-100-150-200-300-500): un valore numerico che rappresenta un'approssimazione del rifornimento proposto. Le classi possibili sono: 20-50-100-150-200-300-500 pezzi.

Il dataset è stato realizzato manualmente, curando l'inserimento di ogni singolo record e facendo in modo di coprire quanti più casi possibili. Sono state analizzate le feature e si sono stabiliti dei criteri per l'inserimento di nuovi elementi, cioè abbiamo analizzato le relazioni tra le feature e le etichette. In particolare, tra questi criteri ci sono: la richiesta di prodotti deve essere maggiore nelle festività e nelle zone residenziali, carne venduta di più in inverno e pesce in estate, vendite elettronica e casa quasi indifferente rispetto alle stagioni, carne e pesce molte più vendite durante le festività. Sono stati curati anche i costi che per FruttaVerdura devono essere generalmente bassi, rispetto ad elettronica che possono anche avere prezzi molto alti. Ovviamente si preferisce acquistare più prodotti che ingombrano di meno il magazzino (dimensione confezione piccola), che scadono il più lontano possibile e la cui spedizione e costo sia minore.

Il dataset proposto è un file *.csv* composto da 760 elementi.

Notebook

Il codice per l'addestramento e l'utilizzo del classificatore è stato proposto inizialmente attraverso un notebook jupyter scritto in python, al fine di valutarne tutti gli aspetti e l'accuratezza. Il classificatore utilizzato è stato importato dalla libreria scikit-learn[1]: RandomForestClassifier[2]

Il lavoro eseguito nel notebook può essere cosi riassunto:

1. caricamento dataset
2. divisione in training set e test set
3. addestramento della Random Forest
4. calcolo dell'accuratezza sui dati di test
5. calcolo dell'importanza di ogni feature e visualizzazione con grafico
6. estrapolazione di un albero dalla Random Forest
7. visione dettagliata di una parte dell'albero
8. traduzione in java

All'inizio andiamo ad importare il dataset grazie alla libreria *pandas,* che ci permette di leggere un file *.csv* e trasformarlo in una tabella.

Successivamente separiamo le features dalle etichette e, visto che il classificatore non accetta stringhe, andiamo a convertire i campi delle features in variabili fittizie, utilizzando la funzione get_dummies. Questo tipo di conversione è chiamato one-hot encode e consiste nel rimuovere la variabile categorica (una stringa) e aggiungere una nuova variabile binaria per ogni valore univoco della variabile categorica. Per esempio: la variabile stagione viene trasformata in 4 variabili (stagione_Inverno, stagione_Estate, stagione_Primavera, stagione_Autunno) e a quella giusta viene assegnata 1 e a tutte le altre 0. Adesso separiamo i dati in training set per l'addestramento e test set per il testing, grazie all'utilizzo della funzione train_test_split di sklearn.

Importiamo RandomForestClassifier e lo istanziamo, settando alcuni parametri utili al classificatore:

- n_estimators=99, impostiamo il numero degli alberi a 99, in considerazione che è preferibile utilizzare un numero dispari
- bootstrap=True, viene utilizzato il metodo di bootstrap in cui abbiamo sottoinsiemi di variabili
- random_state=0, sottoinsiemi di variabili casuali

Lasciamo il default per altri parametri come per esempio il criterio=*gini*, la funzione per misurare la qualità di uno split di un nodo dell'albero. Essa si basa sul concetto di purità, un nodo si dice puro quando tutti i campioni in esso contenuti hanno uguale etichetta. In particolare, l'indice di eterogeneità di Gini offre una misura della eterogeneità a partire dai valori delle frequenze relative associate alle k modalità di una generica variabile X. Ciò vuol dire che se i dati sono distribuiti in modo eterogeneo su tutte le k modalità di X, l'indice di Gini è elevato, viceversa, in caso di distribuzione di frequenza omogenea l'indice sarà piuttosto basso.

Addestriamo il classificatore sul training set e andiamo a predire il test set. Calcoliamo l'accuratezza come somiglianza tra le etichette predette e le etichette del test set e la mostriamo. Considerazioni sui risultati sono descritte nel paragrafo Risultati ottenuti.

Adesso mostriamo tutte le feature ordinate per la loro importanza per il classificatore. Utilizziamo il metodo *series* di *pandas* che ci permette di mostrare i valori di importanza indicizzati per ogni feature. Creiamo un *bar plot* (un grafico a barre) dove mostriamo i risultati ottenuti.

A causa della codifica one-hot la visualizzazione non è ottimale, infatti alcune variabili sono replicate per ogni valore che esse possono assumere. Per questo aggreghiamo le stesse variabili e rifacciamo un altro *bar plot.*

Estraiamo un albero decisionale dal classificatore, lo esportiamo come immagine e lo mostriamo. L'albero risultato è davvero grandissimo e quasi illeggibile a causa della sua complessità. Per questo creiamo un nuovo classificatore limitando la sua profondità al livello 3 ed estraiamo un albero decisionale.

A questo punto sfruttiamo la libreria sklearn_porter[3] che ci offre un Porter in grado di trasformare il classificatore in una classe java, salvandolo su un file. Questo ci servirà per poter integrare il modulo nel nostro prodotto software NewDM, scritto in java.

In allegato il pdf del notebook, dove è presente il codice commentato e tutti i grafici risultanti dai dati per una migliore chiarezza.

Integrazione con il software NewDM

La classe java ottenuta come risultato del Porter è RandomForestClassifier ed è stata salvata all'interno della cartella del software NewDM per utilizzarla. A causa della sua complessità e lunghezza si è dovuto modificare alcuni limiti al nostro *IDE* per utilizzarla. Essa ci fornisce un metodo statico: *predict*, al quale passando l'array degli input, ci restituisce l'etichetta predetta. L'array deve essere composto da tutti i valori delle features, ricordando che alcune features sono state scomposte a causa della codifica one-hot. Il risultato restituito è un intero che corrisponde all'indice dell'etichetta. Per esempio 0 corrisponde a 20 pezzi, 1 a 50 pezzi e così via.

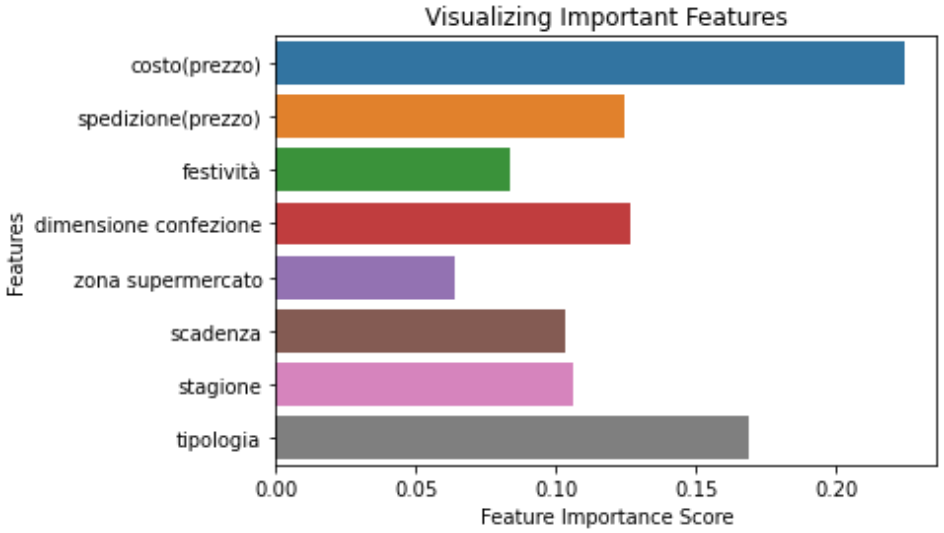
Per questo è stata creata un ulteriore classe che invoca il metodo *predict* e che effettua tutte le dovute conversioni per gli input e gli output.

Il tutto è stato poi integrato con le altre funzionalità del software e sono state create delle interfacce apposite per l'immissione dei parametri e la visione dell'output. Ulteriori informazioni sono esposte in risultati ottenuti.

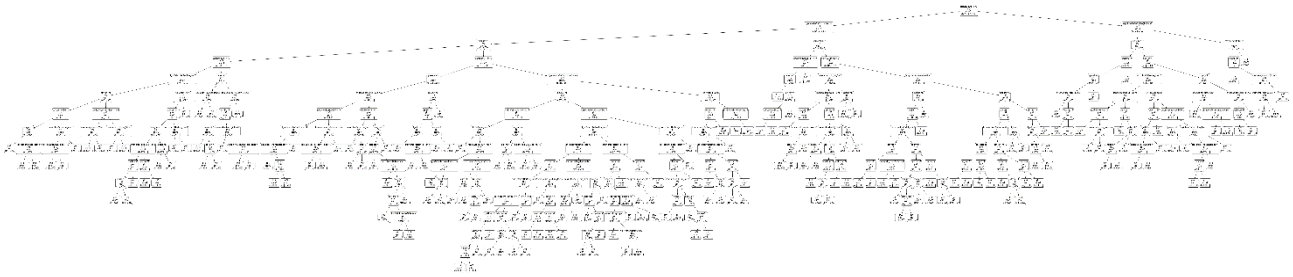
4. Risultati ottenuti

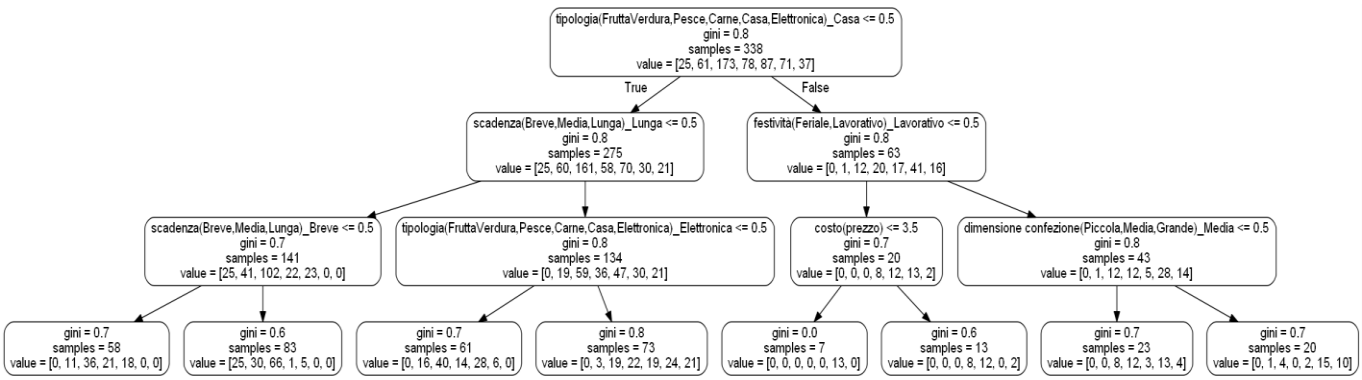
Una volta addestrato il classificatore abbiamo testato più volte l'accuratezza sul test set e si può dire che essa è compresa tra il 75% e l'85%.

Dopo aver ottimizzato la visualizzazione delle importanze delle features e creato il bar plot corrispondente, abbiamo riscontrato che i valori di ognuno sono abbastanza equilibrati e uniformi e che quindi sono tutte importanti per il risultato finale. In particolare, possiamo però notare che il costo del prodotto ha l'*Importance Score* più alto degli altri ed è quindi quello che caratterizza di più la previsione.



Grazie all'estrapolazione di uno dei tanti alberi, abbiamo potuto osservare il reale comportamento del classificatore, per valutare eventuali modifiche dei parametri allo stesso. A causa della sua complessità se ne mostra anche una versione con una profondità ridotta.





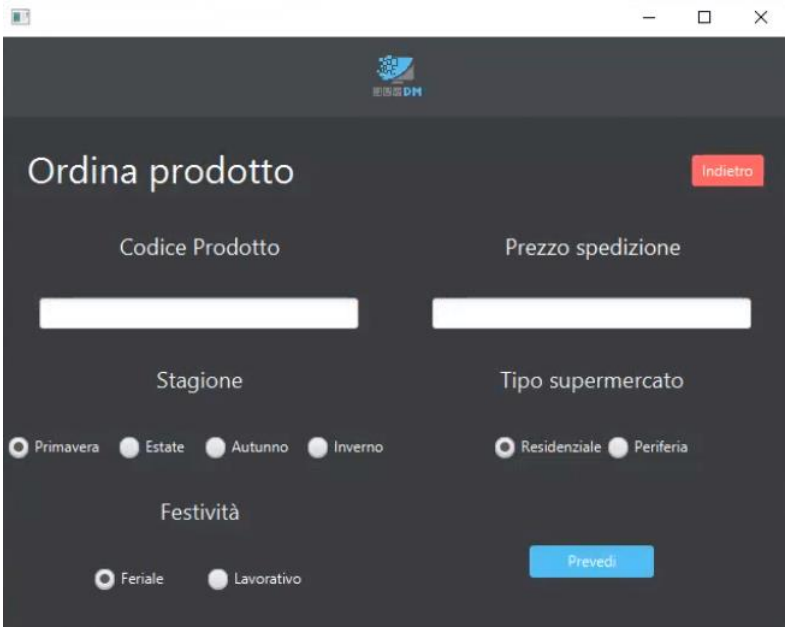
In esso possiamo osservare le caratteristiche, i criteri di splitting e l'effettiva efficienza di splitting basandoci sull'indice di purità *gini*. *Samples* rappresenta il numero di elementi per ogni nodo e *value* è l'array degli elementi raggruppati per etichetta.

Una volta raffinato al meglio il classificatore, grazie al Porter abbiamo ottenuto il codice java, che ci ha permesso di integrarlo con il nostro software. Il magazziniere che utilizzerà questo strumento accederà alla pagina per ordinare prodotti, inserirà i dati e cliccando sul pulsante "Prevedi" otterrà il risultato. I dati da inserire sono:

- Codice prodotto
- Stagione
- Festività
- Prezzo spedizione
- Tipo supermercato

Il software attraverso il codice prodotto risalirà alle features restanti:

- Costo
- Tipologia
- Dimensione confezione
- Scadenza



Una volta ottenuta la previsione potrà confermare l'acquisto oppure come ulteriore funzionalità potrà anche inserire manualmente le quantità da acquistare.

5. Motivazioni delle scelte

Molte motivazioni sono state già espresse nei punti precedenti, durante le spiegazioni degli argomenti stessi. Alcune motivazioni però richiedevano ulteriori conoscenze, esplicate poi nei punti successivi, perciò si è deciso di rimandare ad ulteriori spiegazioni che saranno fornite in questo paragrafo.

Tra queste abbiamo il perché di questo determinato classificatore. La scelta è ricaduta sulla Random Forest perché essa risponde bene alle esigenze della situazione, cioè gestire le risposte relative ad una serie di domande al fine di restituire l'etichetta giusta per i dati in input.

Infatti, questo classificatore si presta molto bene ai problemi decisionali ed è un classificatore n-ario completamente automatizzabile. Lavora bene con valori numerici e categorici e non è influenzato da valori anomali, quindi risponde alle esigenze del nostro dataset.

Le Random Forest generalmente forniscono un'elevata precisione poiché il principio del modello è quello di calcolare la media dei risultati tra i molteplici alberi decisionali che costruisce.

6. Conclusioni e sviluppi futuri

Il modulo creato potrebbe essere davvero di aiuto all'attività dei magazzinieri, magari anche con ulteriori raffinazioni del software. Si potrebbe pensare di alleggerire il compito del dipendente riducendo le informazioni che deve inserire in input, per esempio festività e stagione potrebbero essere campi ottenibili direttamente dalla data corrente. Si potrebbe poi pensare di impostare di default la zona del supermercato, visto che è sempre quella per ogni punto vendita. In questa versione non è stato fatto per dare una maggiore chiarezza del lavoro eseguito dal modulo.

Un ulteriore sviluppo potrebbe essere quello di poter riaddestrare il classificatore con i nuovi dati. Le scelte sarebbero 2: aggiungere ogni volta degli elementi allo stesso dataset migliorandolo, oppure creare un nuovo dataset ad ogni addestramento. In tutti e due i casi il software dovrebbe salvare ogni volta gli elementi in un file per un futuro riaddestramento. Per motivazioni tecnologiche non è stato eseguito in questa prima versione del software, infatti il Porter non consente il riaddestramento con nuovi dati. Si potrebbe pensare quindi di utilizzare un'altra tecnologia per raggiungere questi obiettivi.

Collegamenti esterni

[1]: link: https://scikit-learn.org

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

[3]: link: https://github.com/nok/sklearn-porter

In [1]:
```
#Import dataset da libreria scikit-learn
from sklearn import datasets
import pandas as pd

#Caricamento dataset
data = pd.read_csv("dataset2.csv ", header = 0)
#data.head()
```

In [14]:
```
# Import train_test_split function
from sklearn.model_selection import train_test_split

#Dal mio dataset vado a separare le feature e le classi
y=data["fornitura(20-50-100-150-200-300-500)"]
#converte le stringhe in set di boolean (One-hot encode)
X = pd.get_dummies(data[["tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)","stagione","zona supermercato(Periferia,Residenziale)","festività(Feriale,Lavorativo)","scadenza(Breve,Media,Lunga)","dimensione confezione(Piccola,Media,Grande)","costo(prezzo)","spedizione(prezzo)"]])
#print(X)

# Split dataset nel training set e test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training e 30% test
#print(X_test,y_test)
```

In [15]:
```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Creazione classificatore con alberi dispari
clf=RandomForestClassifier(n_estimators=99)

#bootstrap=vengono utilizzati sottinsieme di variabili, random_state=sottinsiemi di variabili casuali
clf=RandomForestClassifier(n_estimators=99,bootstrap=True,random_state=0)


#Addestramento
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
#print(X_test)

#res=clf.predict([[15,15,1,0,0,0,0,1,1,0,0,0,1,1,0,0,1,0,0,1]])
#print(res)
```

In [16]:
```
#Import scikit-learn metrics module per il calcolo dell'accuracy
from sklearn import metrics
# Model Accuracy, quanto spesso il classificatore è corretto?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
#print(y_test, y_pred)
```

```
Accuracy: 0.8245614035087719
```

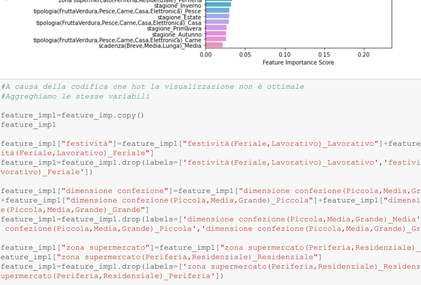
In [17]:
```
#Series rappresenta dati 1D
#Mostriamo l'importanza delle feature ordinate in ordine decrescente
feature_imp = pd.Series(clf.feature_importances_,index=X.columns).sort_values(ascending=False)
feature_imp
```

Out[17]:
```
costo(prezzo)                                                                  0.223911
spedizione(prezzo)                                                             0.115466
dimensione confezione(Piccola,Media,Grande)_Grande                             0.050331
dimensione confezione(Piccola,Media,Grande)_Piccola                            0.049490
dimensione confezione(Piccola,Media,Grande)_Media                              0.046266
tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_FruttaVerdura            0.044518
festività(Feriale,Lavorativo)_Lavorativo                                       0.040242
tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Elettronica              0.037111
festività(Feriale,Lavorativo)_Feriale                                          0.036560
scadenza(Breve,Media,Lunga)_Lunga                                              0.034247
scadenza(Breve,Media,Lunga)_Breve                                              0.033796
zona supermercato(Periferia,Residenziale)_Residenziale                         0.033549
zona supermercato(Periferia,Residenziale)_Periferia                            0.033531
stagione_Inverno                                                               0.031828
tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Pesce                    0.030381
stagione_Estate                                                                0.029753
tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Casa                     0.029467
stagione_Primavera                                                             0.026340
stagione_Autunno                                                               0.025994
tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Carne                    0.025633
scadenza(Breve,Media,Lunga)_Media                                              0.021584
dtype: float64
```

In [18]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Creiamo un bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)

# Aggiungiamo le etichette
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
#plt.legend()
plt.show()
```



In [19]:
```
#A causa della codifica one hot la visualizzazione non è ottimale
#Aggreghiamo le stesse variabili

feature_imp1=feature_imp.copy()
feature_imp1

feature_imp1["festività"]=feature_imp1["festività(Feriale,Lavorativo)_Lavorativo"]+feature_imp1["festività(Feriale,Lavorativo)_Feriale"]
feature_imp1=feature_imp1.drop(labels=['festività(Feriale,Lavorativo)_Lavorativo','festività(Feriale,Lavorativo)_Feriale'])

feature_imp1["dimensione confezione"]=feature_imp1["dimensione confezione(Piccola,Media,Grande)_Media"]+feature_imp1["dimensione confezione(Piccola,Media,Grande)_Piccola"]+feature_imp1["dimensione confezione(Piccola,Media,Grande)_Grande"]
feature_imp1=feature_imp1.drop(labels=['dimensione confezione(Piccola,Media,Grande)_Media','dimensione confezione(Piccola,Media,Grande)_Piccola','dimensione confezione(Piccola,Media,Grande)_Grande'])

feature_imp1["zona supermercato"]=feature_imp1["zona supermercato(Periferia,Residenziale)_Periferia"]+feature_imp1["zona supermercato(Periferia,Residenziale)_Residenziale"]
feature_imp1=feature_imp1.drop(labels=['zona supermercato(Periferia,Residenziale)_Residenziale','zona supermercato(Periferia,Residenziale)_Periferia'])

feature_imp1["scadenza"]=feature_imp1["scadenza(Breve,Media,Lunga)_Media"]+feature_imp1["scadenza(Breve,Media,Lunga)_Lunga"]+feature_imp1["scadenza(Breve,Media,Lunga)_Breve"]
feature_imp1=feature_imp1.drop(labels=['scadenza(Breve,Media,Lunga)_Media','scadenza(Breve,Media,Lunga)_Lunga','scadenza(Breve,Media,Lunga)_Breve'])

feature_imp1["stagione"]=feature_imp1["stagione_Primavera"]+feature_imp1["stagione_Estate"]+feature_imp1["stagione_Autunno"]+feature_imp1["stagione_Inverno"]
feature_imp1=feature_imp1.drop(labels=['stagione_Inverno','stagione_Estate','stagione_Primavera','stagione_Autunno'])

feature_imp1["tipologia"]=feature_imp1["tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Elettronica"]+feature_imp1["tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Carne"]+feature_imp1["tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Pesce"]+feature_imp1["tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_FruttaVerdura"]+feature_imp1["tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Casa"]
feature_imp1=feature_imp1.drop(labels=['tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Elettronica','tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Carne','tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Pesce','tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_FruttaVerdura','tipologia(FruttaVerdura,Pesce,Carne,Casa,Elettronica)_Casa'])


feature_imp1.sort_values(ascending=False)
```

Out[19]:
```
costo(prezzo)             0.223911
tipologia                 0.167111
dimensione confezione     0.146087
spedizione(prezzo)        0.115466
stagione                  0.113915
scadenza                  0.089628
festività                 0.076802
zona supermercato         0.067080
dtype: float64
```

In [20]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Ricreiamo il bar plot
sns.barplot(x=feature_imp1, y=feature_imp1.index)


plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
#plt.legend()
plt.show()
```

In [21]:
```
# Tools per la visualizzazione
from sklearn.tree import export_graphviz
import pydot

# Prendiamo un albero dalla foresta
tree = clf.estimators_[5]

feature_list=list(X.columns)
# Esportiamo l'immagine
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True, precision = 1)

# Usiamo un file dot per salvare l'immagine
(graph, ) = pydot.graph_from_dot_file('tree.dot')

# Trasformiamo in png
graph.write_png('tree.png')
```

In [22]:
```
from IPython.display import Image
Image(filename='tree.png')
```

Out[22]:

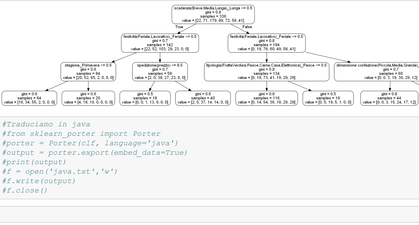
In [23]:
```
# Limitiamo la profondità a 3 livelli
rf_small = RandomForestClassifier(n_estimators=10, max_depth = 3)
rf_small.fit(X_train,y_train)

# Estraiamo l'albero ridotto
tree_small = rf_small.estimators_[5]

# Salviamo come png
export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = feature_list, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('small_tree.dot')
graph.write_png('small_tree.png');

# Mostriamo
Image(filename='small_tree.png')
```

Out[23]:



In [18]:
```
#Traduciamo in java
#from sklearn_porter import Porter
#porter = Porter(clf, language='java')
#output = porter.export(embed_data=True)
#print(output)
#f = open('java.txt','w')
#f.write(output)
#f.close()
```

In [ ]:

In [ ]: