Laurea Triennale in Informatica - Università di Salerno

Corso di Machine Learning

Prof. G. Polese - Prof.ssa L. Caruccio

Lorenzo Castellano

Repository GitHub

Indice

1 Introduzione

In un mondo globalizzato sempre più spesso ci si ritrova a dover considerare posti di cui si è sentito parlare, ma su cui non si hanno informazioni concrete. Quali sono i modi e gli strumenti a disposizione dell'utente che affronta tale valutazione?
Servizi di mappe, forum e blog mettono a disposizione varie informazioni su una determinata area, con una naturale propensione verso le grandi città e i punti di maggiore interesse. Le informazioni sono però sparse e distribuite su varie fonti.

1.1 Sistema Completo

Il modulo proposto si propone di raccogliere all'interno di un unico strumento tutte le funzionalità che possano assistere l'utente in uno "screening" iniziale di una località qualsiasi sul territorio italiano. In particolare sulla base di dati generici e ad ampio spettro il modello riesce a fornire una valutazione oggettiva di città e paesi specifici sulla mappa, ponendosi come strumento fondamentale per la formazione di un'idea prescisa dell'affinità della località cercata ai parametri inseriti.
Il modulo elabora i dati fornendo i risultati sottoforma di un dataset che funge da classifica di 5 località, ordinando i risultati per IdQ e scartando quelli non pertinenti. L'accesso al modulo viene fornito sottoforma di package Python, fornendo la possibilità di ricevere i risultati a seconda dei parametri inseriti all'interno di uno script Python.

2 Descrizione dell'agente

2.1 Obiettivi

Lo scopo del progetto è quindi quello di realizzare un agente intelligente che sia in grado di:

- Fornire un Indice di Qualità in base ai dati dei comuni trovati;
- Consigliare un gruppo di comuni all'interno di un'area indicata che non solo massimizzi l'IdQ, ma rispetti parametri immessi dall'utente.

2.2 Specifica PEAS

La specifica PEAS del nostro modello di Machine Learning è la seguente:

- **Performance**: Le misure di prestazione adottate per valutare l'operato del nostro modello di Machine Learning sono: *Mean Squared Error (MSE)*, *Accuracy*, *Standard Deviation*, *Distribuzione dell' Errore Residuo* e *Dispersione dei Valori Predetti*;
- **Environment**: Gli elementi che costituiscono l'ambiente sono principalmente dati recuperati dall'Istituto Nazionale di Statistica (*ISTAT*) e dalle *API HEREv7* del servizio di mappe *HERE*;
- **Actuators**: Il modello valuterà ogni comune trovato e assegnerà per ognuno un valore indicante l'Indice di Qualità;
- **Sensors**: Il modello riceve gli input necessari e quindi i filtri personalizzati da un utente attraverso l'interfaccia web del sistema.

2.2.1 Caratteristiche dell'ambiente

Le caratteristiche dell'ambiente sono le seguenti:

- **Completamente osservabile**: conosciamo, attraverso vari dataset e API, tutti i dati di un comune di cui abbiamo bisogno per predire l'IdQ;
- **Stocastico**: non sappiamo cosa può succedere all'ambiente nel momento in cui viene effettuata una predizione;
- **Episodico**: ogni misurazione può essere vista come un episodio unico, poiché non c'è correlazione tra le varie misurazioni/predizioni;
- **Statico**: le caratteristiche dell'istanza di input e la relativa predizione rimangono costanti durante la previsione;
- **Continuo**: le caratteristiche dell'ambiente possono assumere un numero infinito di valori all'interno di un determinato intervallo;
- **Singolo**: L'ambiente consente la presenza di un unico agente, utilizzato per predire l'Indice di Qualità.

2.3 Analisi del problema

Si poteva affrontare il problema effettuando una ricerca in un database che contenesse gli indici per ogni comune, trovando quelli che più andassero incontro ai parametri inseriti dall'utente. Tuttavia tale database non è disponibile, soprattutto non in una versione che comprenda tutte le possibili località a cui l'utente medio potrebbe essere interessato, soprattutto per quanto riguarda quelle di minor interesse. Tale database sarebbe inoltre di difficile mantenimento e aggiornamento, dato il grande numero di dati da considerare.
Si è pertanto deciso di affidarsi a un **modello di Machine Learning** che possa fornire risultati calcolati al momento e utilizzando dati aggiornati costantemente, così da tenere traccia di qualsiasi eventuale fluttuazione.
Essendo necessario predire una **variabile continua** (Indice di Qualità), la situazione è stata interpretata come un problema di **regressione**, trattando i dati a disposizione come *variabili indipendenti* e l'IdQ da calcolare come *variabile dipendente*.

3 Tecnologie usate

Il linguaggio scelto per la realizzazione del Modulo è **Python**. Ci si è inoltre avvalsi di diverse API descritte in seguito e del framework **SetupTools**

4 Raccolta, analisi e preprocessing dei dati

4.1 Data Understanding

I dati che sono stati utilizzati per addestrare il modello di Machine Learning provengono da **fonti diverse**. Vista la non esistenza di un dataset adatto alle esigenze del modulo, sono state recuperate tutte le fonti che potessero essere utili e poi creato un dataset che racchiudesse le informazioni necessarie.
In particolare è stata trovata una classifica creata dall'ente "*Il Sole 24 Ore*" in cui ogni provincia è associata a un proprio punteggio di qualità di vita.

	Provincia	Punteggio
0	Udine	605,68
1	Bologna	598,24
2	Trento	597,09
3	Aosta	594,21
4	Bergamo	592,33
...	...	...

Classifica Il Sole 24 Ore

Inoltre, tramite le API dell'*Istituto nazionale di statistica* (*ISTAT*), è stato possibile recuperare le classi di benessere per ogni provincia e tramite queste creare un secondo dataset iniziale in cui

	prov_name	value
0	Agrigento	34,4
1	Barletta-Andria-Trani	22,0
2	Varese	9,8
3	Venezia	11,5
4	Verbano-Cusio-Ossola	11,5
...	...	...

	prov_name	value
0	Agrigento	26,2
1	Barletta-Andria-Trani	13,6
2	Varese	14,8
3	Venezia	14,8
4	Verbano-Cusio-Ossola	19,7
...	...	...

	prov_name	value
0	Agrigento	14,8
1	Barletta-Andria-Trani	37,3
2	Varese	18,0
3	Venezia	19,7
4	Verbano-Cusio-Ossola	18,0
...	...	...

	prov_name	value
0	Agrigento	11,5
1	Barletta-Andria-Trani	16,9
2	Varese	31,1
3	Venezia	29,5
4	Verbano-Cusio-Ossola	24,6
...	...	...

	prov_name	value
0	Agrigento	13,1
1	Barletta-Andria-Trani	10,2
2	Varese	26,2
3	Venezia	24,6
4	Verbano-Cusio-Ossola	26,2
...	...	...

Classi di benessere

ogni provincia possedesse un proprio indice di qualità.

Purtroppo questi due dataset risolvono solo in parte il problema, in quanto non contengono i dati utili all'addestramento del modello.
Molti dei dati necessari al problema vengono recuperati da vari dataset messi a disposizione dalle API dell'ISTAT, in particolare:

- Il costo di vita viene trovato in base alla spesa media per consumi in ogni regione;
- La pericolosità è un dato calcolato in base al numero di denunce per provincia, tasso di criminalità per regione, rischio idrogeologico per comune e tasso di inquinamento per regione, inoltre viene utilizzato anche un dataset provienente dall'Istituto Nazionale di Geofisica e Vulcanologia (INGV) per le zone sismiche in base alle coordinate del comune;
- Il numero di abitanti viene semplicemente trovato in base al comune.

Di seguito viene mostrata la funzione principale usata per il recupero dei dati tramite file JSON scaricati usando le API dell'ISTAT:

```
def trova_valore_per_id(id_istat, request_type):
    if request_type == "inq":
        with open('istatData/inquinamentoRegioni.json', 'r') as file:
            jsonFile = json.load(file)
    elif request_type == "crim":
        with open('istatData/criminalitaRegioni.json', 'r') as file:
            jsonFile = json.load(file)
    elif request_type == "den":
        with open('istatData/denunceProvince.json', 'r') as file:
```

```
            jsonFile = json.load(file)
    elif request_type == "sup":
        with open('istatData/superficieComuni.json', 'r') as file:
            jsonFile = json.load(file)
    elif request_type == "idro":
        with open('istatData/rischioIdrogeologicoComuni.json', 'r') as file:
            jsonFile = json.load(file)
    elif request_type == "pop":
        with open('istatData/popolazione.json', 'r') as file:
            jsonFile = json.load(file)
    elif request_type == "spesa":
        with open('istatData/spesaMediaRegioni.json', 'r') as file:
            jsonFile = json.load(file)

    result = []
    for series in jsonFile['message:GenericData']['message:DataSet']['generic:Series']:
        series_key = series['generic:SeriesKey']['generic:Value']

        if type(id_istat) is not list:
            ref_area_dict = next(
                (i for i in series_key if i['@id'] in ['REF_AREA', 'ITTER107'] and i['@value'] == id_istat),
                None)

            if ref_area_dict:
                try:
                    result.append(float(series['generic:Obs']['generic:ObsValue']['@value']))
                except KeyError:
                    result.append(-1)
        else:
            for id_x in id_istat:
                ref_area_dict = next(
                    (i for i in series_key if i['@id'] in ['REF_AREA', 'ITTER107'] and i['@value'] == id_x), None)

                if ref_area_dict:
                    try:
                        result.append(float(series['generic:Obs']['generic:ObsValue']['@value']))
                    except KeyError:
                        result.append(-1)
                    continue

    if len(result) == 0:
        return -1
    elif len(result) == 1:
        return result[0]
    return result
```

Recupero Dati dai file ISTAT

Per il resto dei dati, che riguardano il numero di *POI* (*Point Of Interest*) per ogni comune, è stato usato un servizio di mappe.

Ne sono stati valutati vari, ogni soluzione presentava i propri vantaggi e svantaggi.
Le soluzioni più conosciute sono risultate non disponibili, data la disponibilità di API esclusivamente a pagamento. Si è dunque considerato OpenStreetMap per recuperare il numero di POI, avendo metodi molto semplici e trattandosi di un progetto open source, c'era però **grande discrepanza** nell'aggiornamento dei dati tra le città più rilevanti e i comuni più piccoli.
Si è quindi optato per l'uso di **HERE v7**, un servizio di mappe con API con una licenza che permetesse di effettuare un numero limitato di chiamate al giorno gratuitamente.
Scelto il servizio, le soluzioni alternative a OpenStreetMap non concedevano metodi per trovare il numero totale di POI in un luogo, ma solo un **numero limitato entro un certo raggio**, per questo si è provveduto a fare una semplice **stima dei POI totali** in base al numero trovato in un raggio ristretto e la superficie del comune:

```
def trova_numero_poi_herev7(comune, poi_type, raggio):
    bounding_box = get_bounding_box(comune, "here")
    coords = trova_coordinate(comune)
    API_KEY = "oY1k3tXVkAI8O68lu62eXTEkuOc7TQb6Pwn2S_ZCXKo"

    if poi_type == 'neg':
        api_url = (f'https://geocode.search.hereapi.com/v1/browse?'
                   f'at={coords[0]},{coords[1]}'
                   f'&in=circle:{coords[0]},{coords[1]};r={raggio}'
                   f'&categories
   =600-6000,600-6100,600-6200,600-6600-0000,600-6700-0000'
                   f'&limit=100&apiKey={API_KEY}')

        response = requests.get(api_url)
        num = None
        if response.status_code == 200:
            num = len(response.json()["items"])
        return num

    elif poi_type == 'rist':
        api_url = (f'https://geocode.search.hereapi.com/v1/browse?'
                   f'at={coords[0]},{coords[1]}'
                   f'&in=circle:{coords[0]},{coords[1]};r={raggio}'
                   f'&categories=100-1000-0000'
                   f'&limit=100&apiKey={API_KEY}')

        response = requests.get(api_url)
        num = None
        if response.status_code == 200:
            num = len(response.json()["items"])
        return num

    elif poi_type == 'scuola':
        api_url = (f'https://geocode.search.hereapi.com/v1/browse?'
                   f'at={coords[0]},{coords[1]}'
                   f'&in=circle:{coords[0]},{coords[1]};r={raggio}'
                   f'&categories=800-8200-0000'
                   f'&limit=100&apiKey={API_KEY}')

        response = requests.get(api_url)
        num = None
```

```
        if response.status_code == 200:
            num = len(response.json()["items"])
        return num

def stima_poi_totali(comune, poi_type, superficie, raggio=None):
    if raggio is None:
        if poi_type == 'neg':
            raggio = 1000
        elif poi_type == 'rist':
            raggio = 200
        elif poi_type == 'scuola':
            raggio = 1000

    num_poi = trova_numero_poi_herev7(comune, poi_type, raggio)
    if num_poi is None or num_poi == 0:
        num_poi = 1
    area_ricerca = (np.pi * raggio * raggio) / (10 ** 6)
    stima = int(superficie / area_ricerca * num_poi)
    return stima
```

Recupero POI HEREv7

Inoltre, utilizzando vari servizi come OpenStreeMap e Here v7 in modo gratuito, vengono **limitate** le richieste che possiamo fare nel tempo, rallentando quindi in modo generale l'ottenimento di tutti i dati per ogni comune.

4.2 Data Preparation

Messi insieme tutti questi dati, si è provveduto alla creazione di un **dataset unico** che associasse gli indici di qualità trovati precedentemente con i vari dati recuperati. Per creare questo dataset, si è dovuto adattare il nome delle provincie ai propri capoluoghi, dovendo scartare anche alcune righe in quanto non adatte al recupero dei dati da parte dell'ISTAT.

Comune	Latitudine	Longitudine	IdQ	Pericolosità	Costo Vita	Abitanti	Superficie	Abitanti per Km2	Num Negozi	Num Ristoranti	Num Scuole
Agrigento	37.3122991	13.57465	32.70582654571532	8.020439566554517	BASSO	55636.0	243.5037	228.4811278021648	620	5813	232
Varese	45.817549	8.8263532	61.21637935478781	14.52850683441388	ALTO	78807.0	54.8393	1437.053354072718	523	16583	122
Venezia	45.4371908	12.3345898	60.84598657678551	27.568633280101608	MEDIO	250913.0	415.8927	603.3118638533449	9796	271384	794
Verbania	45.9344082	8.5580062	56.126538154578974	16.710729643667847	MEDIO	30015.0	37.4934	800.5408951975547	71	298	35
Vercelli	45.3251557	8.4227666	54.58606658572052	11.758912720637149	MEDIO	45399.0	79.7743	569.0930537779711	457	16505	126
Verona	45.4384958	10.9924122	68.859231002828	30.346299420671382	MEDIO	256049.0	198.9134	1287.2385671352458	1013	80728	1329
Vibo Valentia	38.6748544	16.0985276	36.98559836236225	40.9673775650712	BASSO	31177.0	46.5709	669.4523833552712	222	3705	88
Vicenza	45.5488306	11.5478825	63.07476657582775	28.48181363975765	MEDIO	110283.0	80.57979999999999	1368.6184378715263	538	23084	205
Viterbo	42.4168441	12.1051148	48.65055084518614	20.480054100441045	ALTO	66178.0	406.2267	162.90903576746678	3232	142236	775
Belluno	46.1375185	12.2181711	60.69128909338338	31.439472789972687	MEDIO	35549.0	147.2211	241.46674627482065	515	11715	140

Show 10 per page 1 2 3 4 5 6 7 8 9 10 11

Dataset finale

Purtroppo il risultato finale è stato, per mancanza generale di dataset riguardanti l'indice di qualità per ogni comune, un **dataset molto ristretto** con molte variabili indipendenti e per questo va ricercato accuratamente il modello con la migliore accuratezza, cercando di abbassare **l'overfitting** dovuto alla presenza di molteplici variabili.
Si è anche pensato di provare ad aumentare in modo sintetico la quantità di dati disponibili nel dataset, ma questa operazione è sembrata non applicabile in quanto avrebbe richiesto un grande investimento di tempo e non avrebbe comunque portato a buoni risultati rispetto al tempo speso.

5 Data Modeling

5.1 Scelta dell'algoritmo

Avendo scelto di ottenere un singolo valore in forma numerica, si è subito pensato ad un algoritmo di regressione. Ci si è trovati con più feature da considerare variabili indipendenti il che ha portato a varie sperimentazioni con diversi tipi di algoritmi.

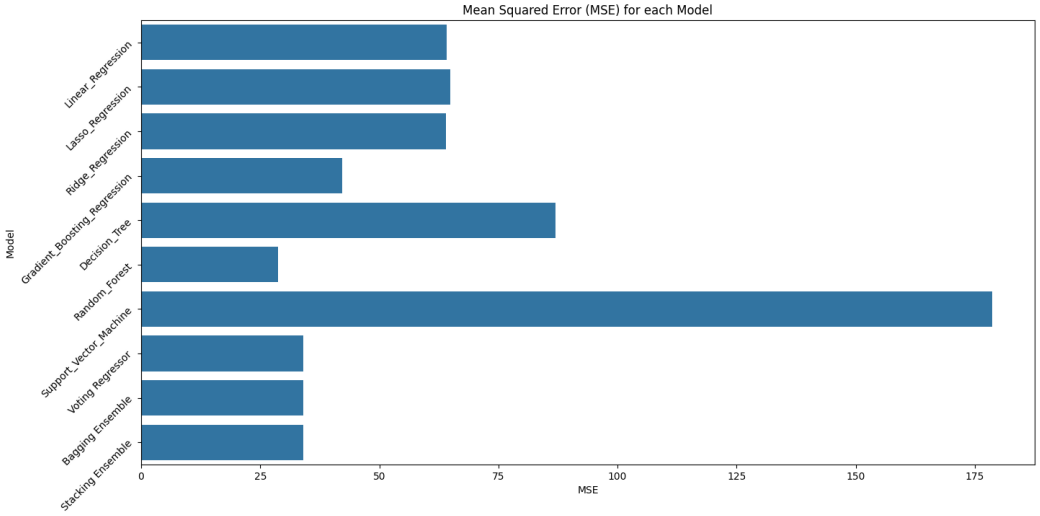
5.1.1 Metrica di valutazione

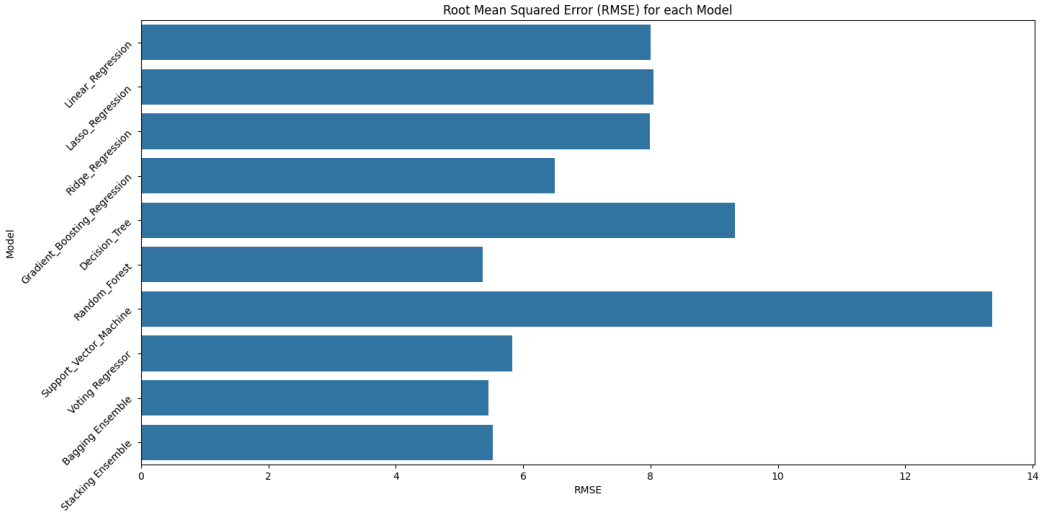
Durante la fase di training, per ricercare il modello migliore da utilizzare ci si è basati su valori come il Mean Squared Error e l'accuratezza trovata tramite il Mean Absolute Percentage Error, inoltre viene fatta anche una K-fold validation con k=5 per ogni modello e viene stampata anche la Deviazione Standard. Vengono addestrati molteplici modelli di regressione e vengono comparati in base ai valori appena descritti.

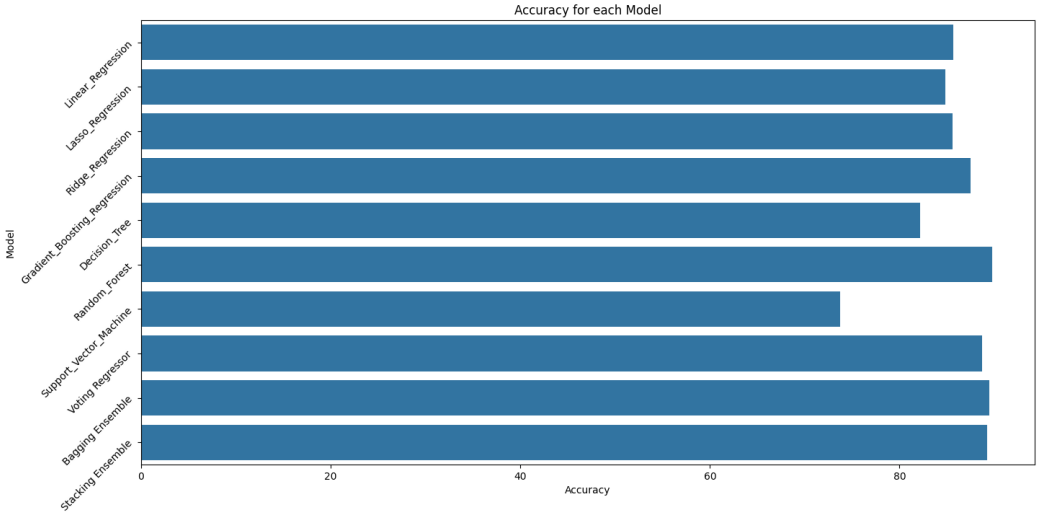
```
models = {
    "Linear_Regression": LinearRegression(),
    "Lasso_Regression": Lasso(),
    "Ridge_Regression": Ridge(),
    "Gradient_Boosting_Regression": GradientBoostingRegressor(random_state=72),
    "Decision_Tree": DecisionTreeRegressor(),
    "Random_Forest": RandomForestRegressor(random_state=59),
    "Support_Vector_Machine": SVR()
}
model_results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = root_mean_squared_error(y_test, y_pred)
    mape = mean_absolute_percentage_error(y_test, y_pred)
    accuracy = round(100*(1 - mape), 2)

    scores = cross_val_score(model, X, y, cv=10, scoring='
   neg_mean_squared_error')
    mse_scores = -scores

    results = {
        'Model': name,
        'MSE': mse,
        'RMSE': rmse,
        'Accuracy': accuracy,
        'Cross Validation MSE': mse_scores.mean(),
        'Cross Validation Standard Deviation': mse_scores.std()
    }

    model_results.append(results)
```

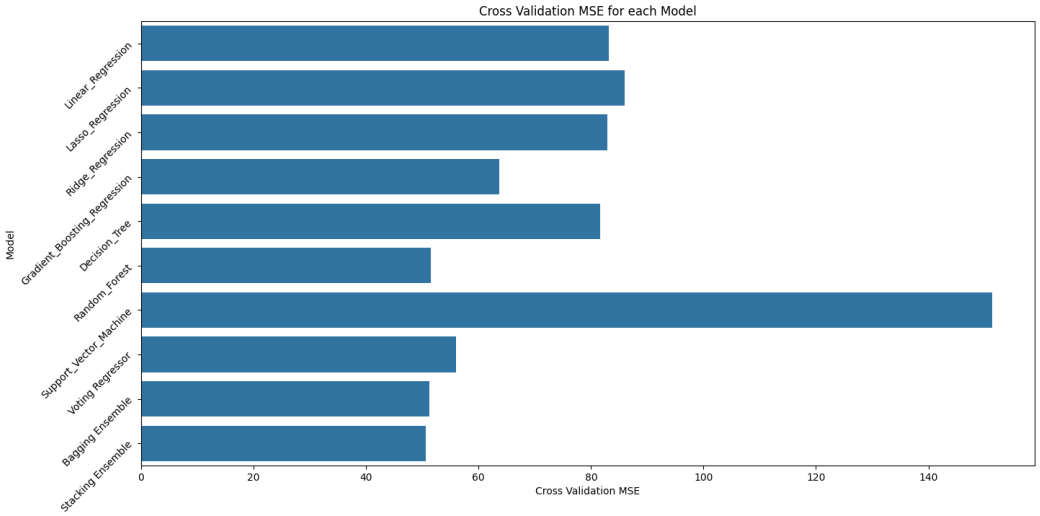
Addestramento Modelli di regressione

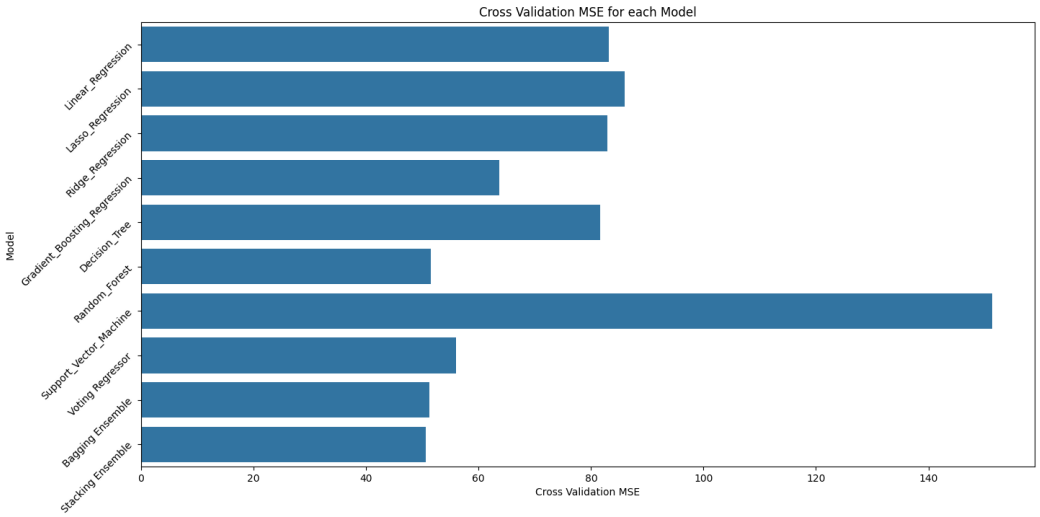
Di seguito vengono mostrati vari grafici che mettono in comparazione i modelli addestrati:






Accuracy for each Model
Model
Linear_Regression
Lasso_Regression
Ridge_Regression
Gradient_Boosting_Regression
Decision_Tree
Random_Forest
Support_Vector_Machine
Voting Regressor
Bagging Ensemble
Stacking Ensemble
0
20
40
60
80
Accuracy


Cross Validation MSE for each Model
Model
Linear_Regression
Lasso_Regression
Ridge_Regression
Gradient_Boosting_Regression
Decision_Tree
Random_Forest
Support_Vector_Machine
Voting Regressor
Bagging Ensemble
Stacking Ensemble
0
20
40
60
80
100
120
140
Cross Validation MSE


Cross Validation MSE for each Model
Model
Linear_Regression
Lasso_Regression
Ridge_Regression
Gradient_Boosting_Regression
Decision_Tree
Random_Forest
Support_Vector_Machine
Voting Regressor
Bagging Ensemble
Stacking Ensemble
0
20
40
60
80
100
120
140
Cross Validation MSE

5.1.2 Random Forest

Messi a confronto i vari dati, notiamo che il modello **Random Forest** per la regressione riporta migliori prestazioni rispetto agli altri, come anche prevedibile in quanto questo modello è adatto a lavorare in casi in cui si ha un numero ristretto di dati e si vuole **ridurre l'overfitting**.
In questo modello vengono addestrati **molteplici alberi decisionali** utilizzando un numero casuale di dati di addestramento e a ogni divisione dell'albero si considera solo un sottoinsieme casuale delle caratteristiche.
Questa casualità nell'addestramento di ciascun albero e nell'utilizzo di diverse caratteristiche permette al modello di essere **meno sensibile all'overfitting e più robusto** rispetto ad un singolo albero decisionale.

5.2 Miglioramento del modello

Dopo aver trovato il modello adatto, si ci è incentrati nel **migliorarlo**, modificando gli iperparametri del modello stesso oppure tramite la creazione di un Ensemble, combinando diversi modelli di base.
Per quanto riguarda la ricerca tramite i migliori iperparametri, questa non restituiva risultati molto diversi rispetto a quelli già avuti di base, anzi sembrava che il modello migliore trovato avesse delle prestazioni un po' peggiori rispetto al Random Forest di base.
Quindi abbiamo spostato il focus sulla creazione di un Ensemble e, dopo vari tentativi, si è giunti all'uso di un **Bagging Ensemble** sullo stesso Random Forest, anche se proprio quest'ultimo risulta essere un Bagging Esemble creato sugli alberi decisionali. Il motivo di questa scelta è che questo modello sembra restituire **prestazioni più stabili** rispetto al solo uso del Random Forest, il quale durante i vari addestramenti riportava valori leggermente diversi ogni volta.

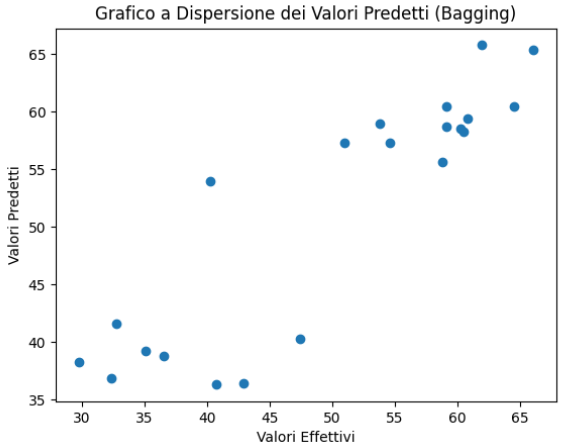
```
bagging_model = BaggingRegressor(estimator=RandomForestRegressor(random_state
    =59), n_estimators=15, random_state=73)
bagging_model.fit(X_train, y_train)
y_pred_bagging = bagging_model.predict(X_test)
mse_bagging = mean_squared_error(y_test, y_pred_bagging)
rmse = root_mean_squared_error(y_test, y_pred_bagging)
mape = mean_absolute_percentage_error(y_test, y_pred_bagging)
accuracy = round(100*(1 - mape), 2)
scores = cross_val_score(bagging_model, X, y, cv=10, scoring='
    neg_mean_squared_error')
mse_scores = -scores
print(f"Mean Squared Error (Bagging) - Mean: {mse_scores.mean()}, Standard
    Deviation: {mse_scores.std()}")

results = {
    'Model': "Bagging Ensemble",
    'MSE': mse,
    'RMSE': rmse,
    'Accuracy': accuracy,
    'Cross Validation MSE': mse_scores.mean(),
    'Cross Validation Standard Deviation': mse_scores.std()
}
model_results.append(results)
```

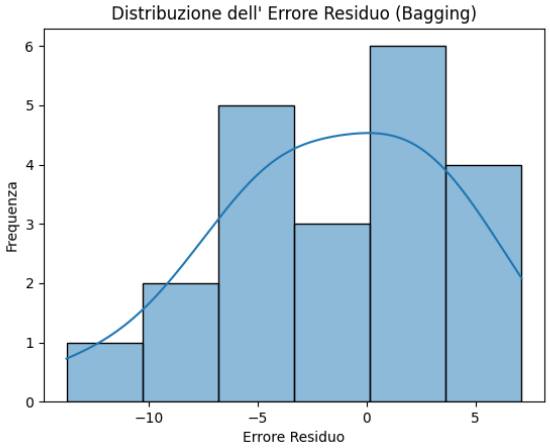
Addestramento Bagging Model

Vengono anche creati dei **grafici** per mostrare quanto il modello sia adatto rispetto ai dati predetti, oltre a fornire l'importanza di ogni variabile indipendente in ogni predizione:

```
plt.scatter(y_test, y_pred_bagging)
plt.xlabel("Valori Effettivi")
plt.ylabel("Valori Predetti")
plt.title(f"Grafico a Dispersione dei Valori Predetti (Bagging)")
plt.show()

residuals = y_test - y_pred_bagging
sns.histplot(residuals, kde=True)
plt.xlabel("Errore Residuo")
plt.ylabel("Frequenza")
plt.title("Distribuzione dell' Errore Residuo (Bagging)")
plt.show()

importance_scores = []
for estimator in bagging_model.estimators_:
    importance_scores.append(estimator.feature_importances_)
average_importance = np.mean(importance_scores, axis=0)
for feature, importance in zip(X_train.columns, average_importance):
    print(f"Feature: {feature}, Importance: {importance}")
```

Addestramento modello






```
Feature: Pericolosità, Importance: 0.0666257185939158
Feature: Costo Vita, Importance: 0.638619470205187
Feature: Abitanti per Km2, Importance: 0.07566294423962219
Feature: Num Negozi Km2, Importance: 0.03908074568849025
Feature: Num Ristoranti Km2, Importance: 0.08564247755983416
Feature: Num Scuole Km2, Importance: 0.09436864371295067
```

Come si può notare, il **Costo Vita** possiede un'importanza notevolmente più alta rispetto alle altre features, creando un limite su quanto un modello possa giudicare senza basarsi troppo su quel particolare dato. Purtroppo non è stato possibile migliorare la situazione, non avendo a disposizione altre features da poter usare nell'addestramento del modello.
Nonostante ciò si può notare dai grafici come il modello sia **abbastanza preciso** per il problema che vogliamo affrontare, anche se risulta ovviamente non perfetto, tendendo a sovrastimare i dati.

6 Utilizzo del modulo

Non essendo disponibile un'interfaccia si è scelto di rendere fruibile il modello attraverso l'utilizzo di un package Python, rendendo il prodotto anche portable e adatto a qualsiasi sistema esegua un ambiente Python funzionante.

6.1 Costruzione package

Per la creazione del package è stato usato **setuptools**, un framework per la distribuzione di pacchetti Python. Questo approccio offre una struttura organizzativa chiara e rende più semplice la gestione delle dipendenze e la distribuzione del software, facilitando anche l'integrazione con il resto del sistema.
Il progetto è strutturato secondo le linee guida di setuptools e segue una convenzione di organizzazione dei file ben definita. La **struttura del progetto** è la seguente:

```
|- Package\
|   |- build\
|   |- dist\
|   |- fyp.a_pkg.egg-info\
|   \- fyp-pkg\
|       |- Database\
|       |- __init__.py
|       |- EstrazioneDatiUsable.py
|       |- InserimentoDati.py
|       |- main.py
|       \- ModelloML.py
|
|- LICENSE
|- README.md
|- setup.py
\- requirements.txt
```

La cartella Package contiene tutto il necessario alla costruzione e al funzionamento del modulo. Viene inoltre fornita la cartella **dist**, contenente le **distribuzioni installabili** del pacchetto.
I moduli specifici del progetto, sono contenuti nella sottocartella **fyp_pkg/**.
All'interno della cartella **Database** è possibile trovare tutti i file non .py utilizzati dal modulo.
Il file **setup.py** definisce le informazioni di distribuzione del pacchetto e le dipendenze necessarie per l'installazione.
Il file **requirements.txt** elenca le dipendenze esterne necessarie per il progetto.
Il file **main.py** esegue un test con dati fittizzi per verificare che tutto funzioni e sia installato correttamente.

7 Conclusioni

Vi sono delle ultime **considerazioni** da fare:
Il modello ottenuto risulta essere sub-ottimo, in quanto vi sono apparenti limitazioni dovute ai dati disponibili e la loro importanza nell'addestramento del modello, ma nonostante ciò ci si può ritenere soddisfatti del lavoro svolto.
Si è cercato di applicare più tecniche utili all'ottenimento, anche se di poco, di un modello migliore, come ad esempio la creazione di un Ensemble e/o la ricerca dei migliori iperparametri.
Volendo migliorare il lavoro svolto, sicuramente sarebbe necessario avere un dataset molto più ampio, con dati etichettati per ogni comune, purtroppo questo risulta essere fuori dalle possibilità del progetto, in quanto richiederebbe una ricerca di dati molto più avanzata e/o la creazione e accumolo di dati per ogni comune.
In generale però, dato il tempo limitato e il tempo totale impiegato nella realizzazione di questo progetto, ci si ritiene più che soddisfatti del risultato finale, avendo acquisito molte esperienze utili nel campo dello sviluppo di un modello di machine learning e l'utilizzo di dati tramite API.

8 Glossario

IdQ (Indice di Qualità di Vita): Indice finale calcolato dal modulo IA.
API (Application Programming Interface): Application Programming Interface. In termini semplici, un API è un insieme di regole e definizioni che consente a diversi software di comunicare tra loro..
Regressione: Tecnica statistica utilizzata per esplorare e modellare la relazione tra una variabile dipendentee e una o più variabili indipendenti. L'obiettivo è quello di comprendere e predire il comportamento della variabile dipendente in base alle variazioni delle variabili indipendenti.
Setuptools: Setuptools è un framework per la distribuzione di pacchetti Python che semplifica la creazione, la distribuzione e l'installazione di software Python.
Framework: insieme di librerie, strumenti e linee guida che forniscono una struttura per lo sviluppo di applicazioni.
Variabili Indipendenti e Dipendenti: le variabili indipendenti sono quelle che sono manipolate o controllate dallo sperimentatore e che si suppone influenzino il risultato, mentre le variabili dipendenti sono quelle che vengono misurate o osservate e che si presume siano influenzate dalle variabili indipendenti.
Dataset: insieme di dati strutturati che vengono utilizzati per scopi di analisi o sperimentazione.
Package e Moduli: I moduli e i package aiutano a organizzare e strutturare il codice in unità più gestibili e riutilizzabili. I moduli contengono funzioni e classi, i package sono un insieme di moduli.