Progetto ICON 21/22

Francesco Bottalico - 718330
f.bottalico15@outlook.it

Contents

1 Progetto

Introduzione. Lo scopo di questo progetto è costruire dei modelli di regressione per la predizione di prezzi di alcune case, partendo da caratteristiche della casa e del luogo dove si trova. Inoltre vengono confrontate le performance di modelli allenati sul dataset originale e su un dataset esteso, al quale sono state aggiunte alcune features attraverso query a **DBpedia**, quindi si vedrà se è possibile migliorare le predizioni dei prezzi attraverso l'integrazione di conoscenza proveniente dal web semantico.

1.1 Requisiti

Il progetto è stato scritto interamente in **Python 3** e sono state usate le seguenti librerie:

- numpy
- scikit-learn
- pandas
- sparqlwrapper

1.2 Files

La repository è organizzata nel seguente modo:

- **data:** in questa cartella sono contenuti i files .csv contenenti il dataset (originale ed esteso)
- **src:** in questa cartella è presente il codice sorgente del progetto, in particolare:
  - **datasparql.py:** contiene il codice per effettuare le query a DBpedia
  - **data_preprocessing.py:** contiene il codice relativo al processing dei dati e per la gestione del dataset
  - **models.py:** contiene il codice relativo alla costruzione dei vari modelli di apprendimento e gestisce il training ed il testing
- **docs:** contiene la documentazione del progetto
- **img:** contiene la documentazione del progetto

2 Analisi dei dati, processing e conoscenza dal web semantico

Nel dataset originale (presente in *data/data.csv*), sono presenti i seguenti attributi:

1. **date:** data di rilevazione
2. **price:** prezzo della casa
3. **bedrooms:** numero di stanze da letto
4. **bathrooms:** numero di bagni
5. **sqft_living:** dimensione della casa
6. **sqft_lot:** dimensione del terreno
7. **floors:** numero di piani
8. **waterfront:** se la casa è sul mare
9. **view:** voto della vista
10. **condition:** condizione della casa
11. **sqft_above:** dimensione della casa escluse stanze sotterranee
12. **sqft_basement:** dimensione di un eventuale piano sotterraneo
13. **yr_built:** anno di costruzione
14. **yr_renovated:** anno di ristrutturazione
15. **street:** via
16. **city:** città
17. **statezip:** codice zip
18. **country:** nazione

2.1 Estensione del dataset

Nel file *data/ext_data.csv* è presente il dataset che è stato esteso attraverso le query a DBpedia, le quali vengono effettuate nel modulo **datasparql.py**, in particolare la funzione:

```
def citiesDensity(cities: list)
```

permette di estrarre la densità abitativa di una lista di città. Invece la funzione:

```
def citiesCoords(cities: list)
```

permette di estrarre latitudine e longitudine delle città, questo permetterà di sostituire la feature **city** con le features **lat** e **long** e quindi rappresentare la città non come una categoria ma attraverso valori numerici. Un altro vantaggio è che la predizione per una casa in una città che non è presente nel set di training sarà più precisa grazie all'uso di latitudine e longitudine.
La conoscenza proveniente dal web semantico permette quindi di aggiungere le seguenti features:

- **lat:** latitudine
- **long:** longitudine
- **density:** densità di popolazione

La posizione della casa effettivamente influisce sul prezzo, mentre la densità sembra influire meno, come si può vedere nella figura 2.1.

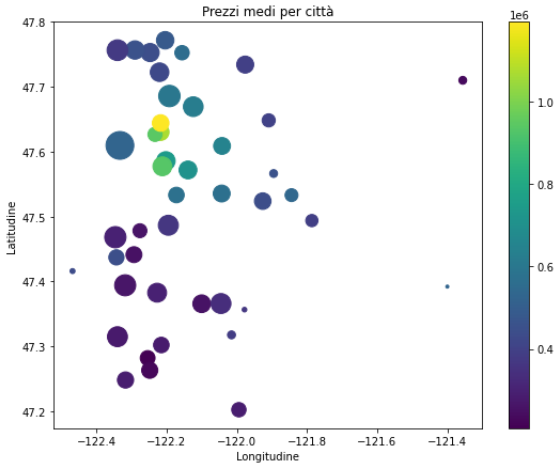

Figure 1: Case raggruppate per città, il colore rappresenta il prezzo medio delle case, la grandezza del punto mostra la densità di popolazione.

2.2 La classe Dataset

Nel modulo **data_preprocessing.py** sono presenti classi e metodi relativi alla gestione del dataset, cioè caricamento da file system, preprocessing e generazione dei set di training e testing.

La classe **Dataset** permette il caricamento del dataset da file system e prova a caricare la relativa versione estesa (chiamata *ext_data.csv*), se questa non viene trovata, allora vengono utilizzati i metodi presenti nel modulo **datasparql.py** per effettuare le query e generare il dataset esteso. Entrambe le versioni del dataset vengono esposte come attributi di classe, rispettivamente denominate **original_data** e **extended_data**.

2.3 Gestione delle features

Sul dataset vengono effettuate diverse operazioni:

Elementi con prezzo nullo Sono presenti 49 entry con prezzo uguale a zero, le quali devono essere rimosse.

```
In [1]: data.price.value_counts()
Out[1]:
0.0          49
300000.0     42
400000.0     31
440000.0     29
450000.0     29
..
684680.0      1
609900.0      1
1635000.0     1
1339000.0     1
220600.0      1
Name: price, Length: 1741, dtype: int64
In [2]: data.drop(index=data[data.price == 0].index, inplace=True)
```

Feature selection Alcune features devono essere rimosse, in particolare:

date La data di rilevazione viene rimossa in quanto non da alcuna informazione aggiuntiva, infatti tutte le date sono ristrette ad un breve periodo di tempo.

street L'indirizzo della casa viene rimosso.

country La nazione viene rimossa in quanto tutte le case presenti nel dataset si trovano negli USA.

waterfront Solo 33 case su 4600 hanno questa feature impostata su 1, quindi l'attributo viene rimosso.

Tracciando la matrice di correlazione tra le feature e il prezzo della casa, e rimuovendo quelle poco correlate si hanno peggioramenti nella predizione, quindi non è stata rimossa nessun altra feature.

Anno di ristrutturazione la feature **yr_renovated** indica l'anno di ristrutturazione della casa, se questa non è mai stata ristrutturata il valore sarà uguale a zero. Per evitare di avere valori nulli, questo valore viene sostituito con il valore della feature **yr_built**, cioé $yr\_renovated = yr\_built$ se la casa non è mai stata ristrutturata.

```
In [3]: data.yr_renovated = data[["yr_built", "yr_renovated"]].max(axis=1)
```

Rimozione outliers Vengono rimossi tutti gli elementi del dataset che contengono almeno un outlier in una feature, cioè se lo zscore di un attributo è maggiore di 2.5 o minore di -2.5 (sono escluse le feature **lat**, **long** e **density**)

```
In [4]: data[(abs(zscore(data.loc[:, :"yr_renovated"])) < 2.5).all(axis=1)]
```

Standardizzazione Alcuni modelli di regressione che verranno utilizzati per effettuare le predizioni sul prezzo funzionano meglio se gli elementi del dataset vengono standardizzati, cioè se i valori delle feature vengono trasformati nei valori di una distribuzione normale standard. Questa operazione viene effettuata prima del training dei modelli nel modulo **models.py**, grazie alla classe **StandardScaler()** della libreria **sklearn.preprocessing**

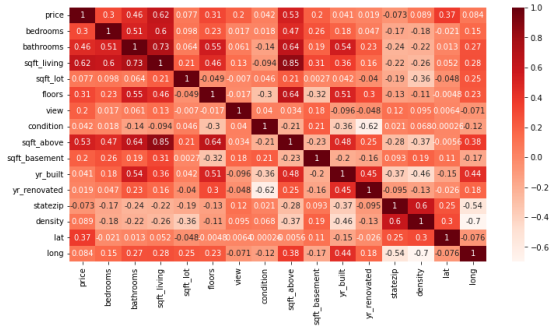

Figure 2: Matrice di correlazione tra le features

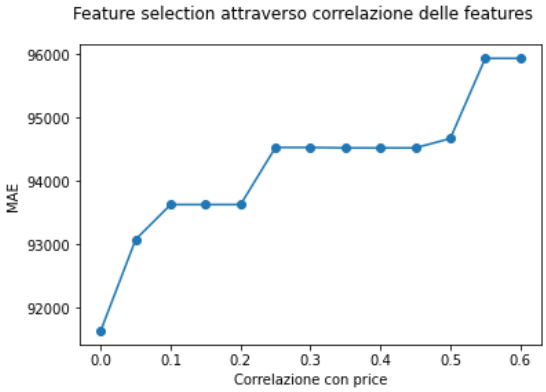

Figure 3: Andamento dell'errore assoluto medio delle predizioni alla rimozione di feature con correlazione minore di un dato valore

One hot encoding Eventuali feature categoriche vengono codificate utilizzando il one hot encoding (come per la standardizzazione questa operazione viene effettuata nel modulo **models.py** prima del training.

Tutte queste operazioni vengono eseguite dalla funzione:

```
def _data_preprocess(data, statezip=False)
```

2.4 Splitting del dataset

Sono infine presenti due metodi per generare il set di training e di testing a partire dal dataset completo.

2.4.1 Equal splitting

La prima funzione è un wrapper per la funzione di splitting **train_test_split()** del modulo **sklearn.model_selection**, che prima di effettuare lo splitting esegue il preprocessing dei dati.

```
def train_test_equal_split(data, tr_size=0.8, ext=False, random_state=None)
```

data Dataset sul quale effettuare lo splitting.

tr_size Dimensione del set di training.

ext True se viene passato il dataset esteso, permette di eliminare la feature **city** che sarà sostituita da **lat**, **long** e **density**.

random_state Per risultati riproducibili.

2.4.2 Train e test con città differenti

La seconda funzione permette di simulare il caso in cui nel set di testing siano presenti città differenti rispetto al set di training, per valutare le performance dei vari modelli di regressione in questo scenario e quindi vedere se l'integrazione della conoscenza dal web semantico permette la creazione di modelli più performanti.

```
def train_test_diffcities_split(data, tr_size=0.8, ext=False, random_state=None)
```

3 Modelli di regressione

Per effettuare le predizioni sul prezzo delle case sono stati usati diversi modelli di regressione lineare e non, al fine di confrontare i risultati dei vari modelli e trovare i più performanti. Questi modelli sono creati, allenati e testati nel modulo **models.py**, dove viene anche effettuata la model selection per trovare dei valori ottimali per gli iperparametri, attraverso una K-fold cross validation ($k = 5$). Ciascun modello viene allenato e testato quattro volte:

1. La prima volta i modelli sono allenati sul dataset originale, il quale viene diviso in training e testing set utilizzando la funzione **train_test_equal_split()**.
2. La seconda volta si utilizza il dataset esteso utilizzando la medesima funzione di splitting. A questo punto sarà possibile confrontare i risultati delle predizioni allenando uno stesso modello allenato però con i diversi dataset.
3. Successivamente i modelli vengono allenati con il dataset originale simulando la situazione in cui nel testing vengono incontrati case in città mai incontrate durante il training (viene utilizzata la funzione **train_test_diffcities_split()**).
4. Infine utilizzando la funzione di splitting precedente, i modelli sono allenati sul dataset esteso, così da confrontare le performance tra dataset originale ed esteso in questo scenario.

Per ottenere risultati confrontabili in ogni testing, viene passato lo stesso seed al parametro **random_state** ad ogni chiamata della funzione di splitting, così da ottenere la medesima divisione del dataset.

3.1 Apprendimento Supervisionato

Per primi vengono allenati dei modelli (lineari e non) che utilizzando algoritmi di apprendimento supervisionato, in particolare:

- Rete neurale (multilayer perceptron)
- Regressione lineare con regolarizzazione L1
- Regressione lineare con regolarizzazione L2
- Decision tree
- Random forest
- Support vector machine

Per ciascuno di questi modelli vengono cercati degli iperparametri ottimali utilizzando la classe **GridSearchCV()** fornita dal modulo **sklearn.model_selection**. La model selection, il training ed il testing di ciascun modello viene effettuato dalla funzione **train_eval()**.

3.1.1 Rete neurale

Per la rete neurale viene effettuata la model selection su diversi parametri quali: algoritmo di ottimizzazione, numero di neuroni e strati e parametro per la regolarizzazione L2 (α nella formula 1). La funzione 1 mostra il valore che la rete neurale cerca di ottimizzare.

$$Loss(y, \hat{y}, W) = \frac{1}{2}(\hat{y} - y)^2 + \alpha \|W\|_2^2 \tag{1}$$

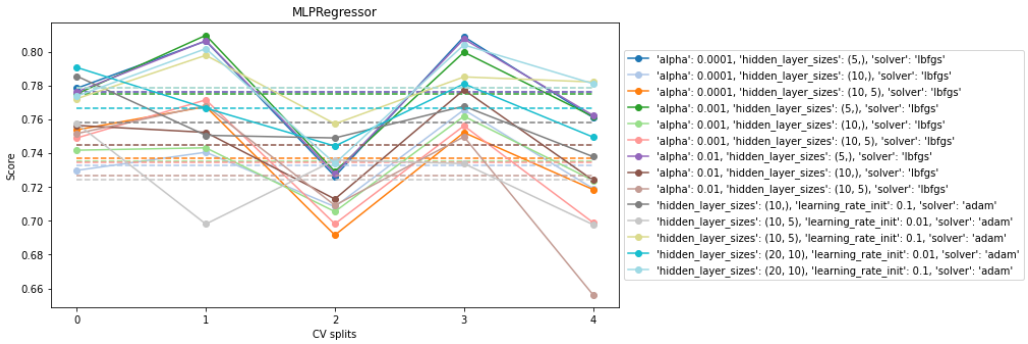

Figure 4: Andamento dello score sui vari fold della cross validation dati diversi valori per gli iperparametri. le rette tratteggiate indicano il valore medio.

3.1.2 Regressione lineare

Gli iperparametri che vengono stimati dalla cross validation sono il learning rate, il suo metodo di aggiornamento ed il parametro di regolarizzazione α.

Regolarizzazione L1 In questo caso viene utilizzata la discesa di gradiente stocastica per ottimizzare la funzione 2.

$$Loss(y, \hat{y}, W) = \frac{1}{2}(\hat{y} - y)^2 + \alpha \|W\| \tag{2}$$

Regolarizzazione L2 La funzione che viene ottimizzata in questo caso è la stessa della rete neurale (formula 1).

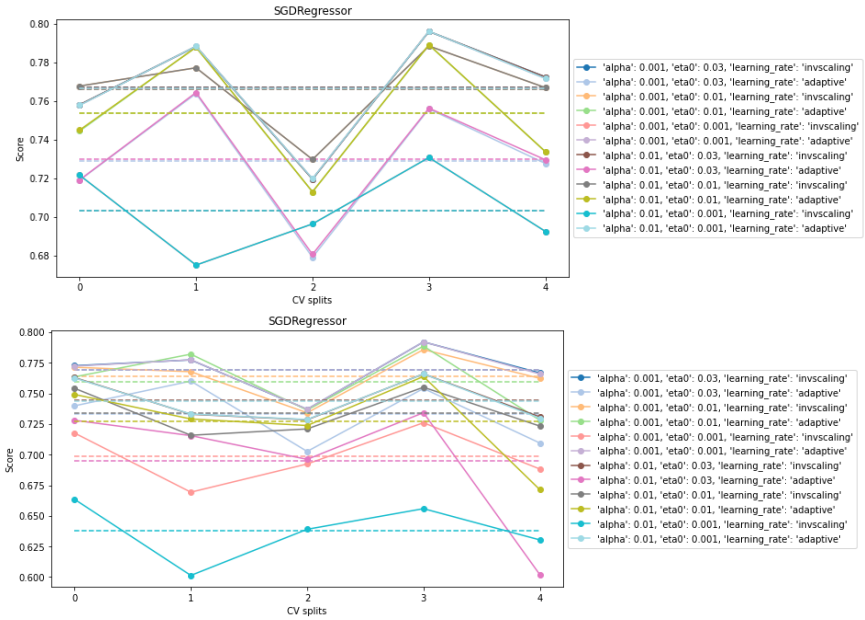

Figure 5: Score delle cross validation per la model selection dei modelli di regressione lineare con regolarizzazione L1 (sopra) ed L2 (sotto).

3.1.3 Decision tree

Per il decision tree la cross validation viene utilizzata per stimare parametri quali la massima profondità dell'albero ed il numero minimo di sample che una foglia può contenere.

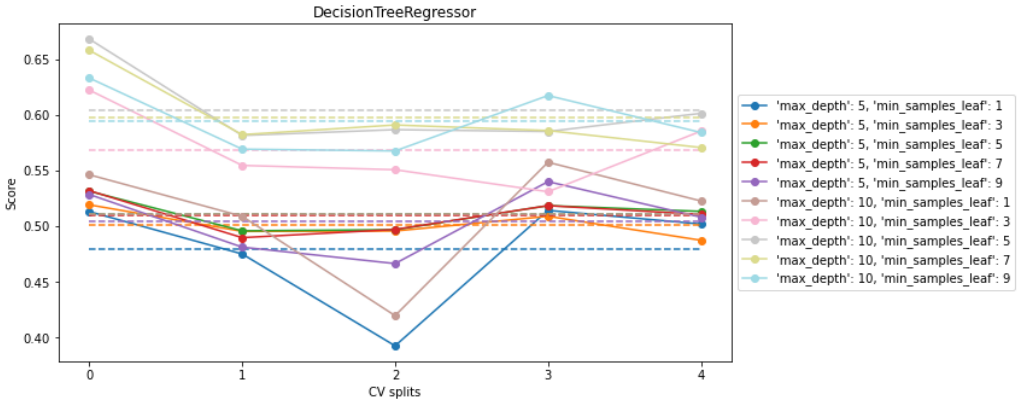

Figure 6: Score delle cross validation per la model selection del decision tree.

3.1.4 Random forest

Per la random forest vengono allenati 100 decision trees, gli iperparametri stimati attraverso la cross validation sono la profondità massima degli alberi ed il numero massimo di feature prese in considerazione durante il training.

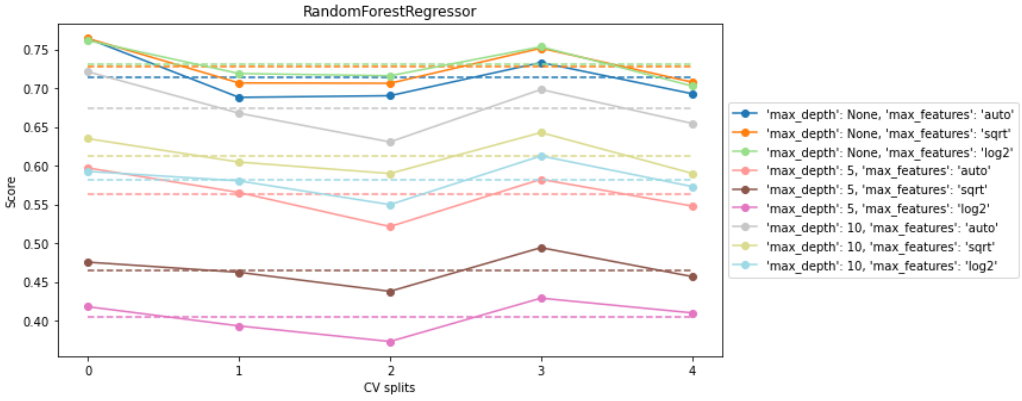

Figure 7: Score delle cross validation per la model selection del decision tree.

3.1.5 Support vector machine

Nel caso della SVM vengono provate diverse funzioni kernel (per il kernel polinomiale si provano diversi gradi) e alcuni valori per il parametro per la regolarizzazione C (che è inversamente proporzionale alla forza della regolarizzazione).

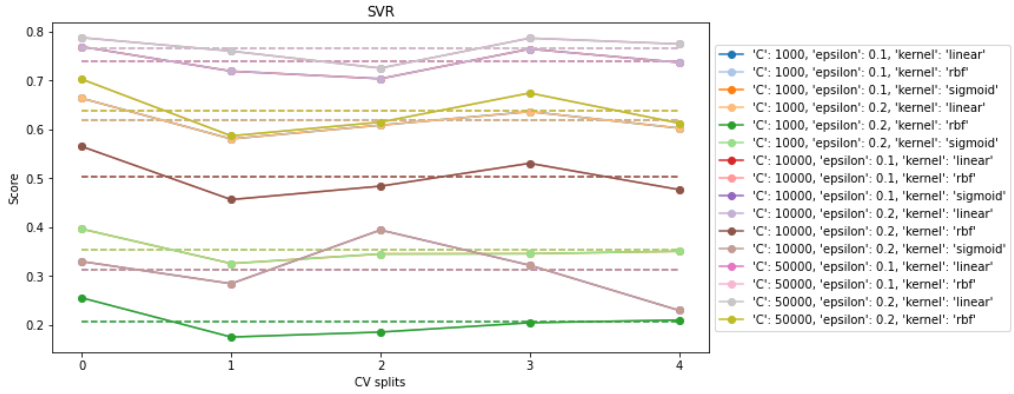

Figure 8: Score delle cross validation per la model selection della support vector machine.

3.2 Clustering

Possono essere usati algoritmi di apprendimento non supervisionato come il K-means, combinati con i precedenti modelli, per migliorare ulteriormente le performance.
L'idea è quella di dividere il training set in cluster e di allenare su ogni cluster un modello supervisionato indipendente da quello degli altri cluster. Per effettuare una predizione il dato viene prima assegnato ad un cluster, per poi utilizzare il relativo modello supervisionato. Il clustering può essere effettuato in due modi:

1. Il K-means viene allenato su tutte le feature dei dati, anche se questi hanno un'alta dimensionalità. In questo caso il miglior valore per k sembra essere $k = 2$ come si vede nella figura a sinistra, anche se i dati non sono molto divisibili e quindi questo approccio probabilmente non migliorerà molto la precisione dei modelli.

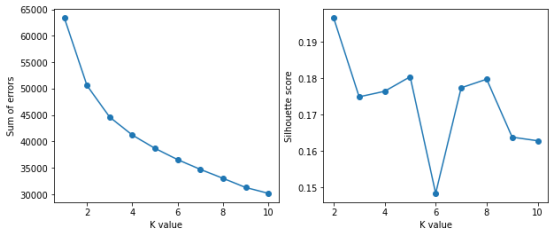

Figure 9: Andamento dell'errore all'aumentare di k nel caso (1)

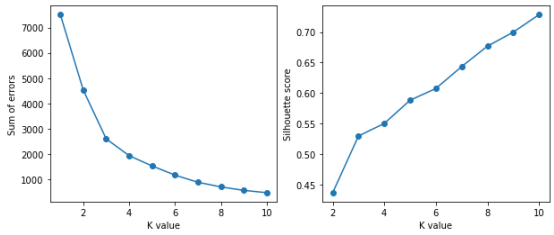

Figure 10: Andamento dell'errore all'aumentare di k nel caso (2)

2. Il K-means viene allenato utilizzando solo le feature latitudine e longitudine (in caso di utilizzo del dataset esteso). In questo caso si prende in considerazione il rapporto che esiste tra luogo della casa ed il suo prezzo (come visto nella figura 2.1 a pagina 3). In questo caso il valore ottimale per k sembra essere $k = 3$ come si può vedere dall'andamento delle curve nella figura a destra.

La seguente funzione divide il training set in k cluster e su ciascuno di questi esegue il training di ogni modello passato in input (viene creata una copia per cluster), inoltre esegue il testing assegnando il sample al relativo cluster ed utilizza per la predizione il relativo modello.

```
def train_eval_cluster(models, train, test, preproc, k, models_per_cl=False,
                       latlong=True, draw=False):
```

models Lista dei modelli da allenare, viene effettuatuata una copia per ogni cluster cosi da avere ogni modello allenato su ogni cluster.

train Dataset di training

test Dataset di testing

preproc Funzione per trasformare i dati prima del training.

k Numero di cluster per il K-means

models_per_cl Se impostato su **True** il parametro models deve essera una lista di k modelli, l'i-esimo modello della lista verrà utilizzato sull'i-esimo cluster. Questa opzione può essere utile per combinare modelli differenti ed avere in alcuni casi prestazioni migliori.

latlong Se applicare il K-means su tutte le feature o solamente su latitudine e longitudine.

draw Se mostrare i grafici del clustering.

4 Confronto dei modelli

Come spiegato nella sezione 3, i modelli verranno allenati quattro volte, per poter confrontare le differenze di prestazioni tra i modelli allenati sul dataset originale e quelli allenati sul dataset esteso, nei due scenari già spiegati.

4.1 Equal splitting

Di seguito sono confrontati i risultati dei vari modelli allenati prima sul dataset originale e poi su quello esteso, utilizzando come funzione di splitting del dataset il metodo **train_test_equal_split()** (spiegato nella sezione 2.4.1). Tra gli stimatori utilizzati, oltre a quelli presentati in precedenza, è stato aggiunto uno stimatore *Dummy* (che predice sempre il valore medio) per confronto.

	Original data			Extended data			
	R2	RMSE	MAE	R2	RMSE	MAE	decr. MAE
Dummy	-0.003	253592.36	191642.73	-0.003	253592.36	191642.73	**0**%
Rete neurale	0.835	102802.54	67716.71	0.843	100257.86	66375.14	**2.021**%
Lasso	0.82	107530.43	71892.61	0.818	107884.67	72514.84	**-0.858**%
Ridge	0.811	109967.49	73629.67	0.812	109665.55	72766.01	**1.187**%
Decision tree	0.688	141346.1	100310.09	0.711	136167.39	91516.67	**9.609**%
Random forest	0.772	120975.81	77201.7	0.785	117388.42	74131.08	**4.142**%
SVM	0.8	113336.14	70691.88	0.796	114286.55	71563.63	**-1.218**%

Table 1: Risultati predizioni usando la funzione di equal splitting.

Non si notano sostanziali miglioramenti allenando i modelli con il dataset esteso, solamente le predizioni effettuate con stimatori basati su alberi di decisione subiscono un buon decremento del *mean average error*, in particolare il decision tree ha un miglioramento di quasi il 10%. Questo accade probabilmente perchè le feature introdotte nel dataset esteso permettono di splittare meglio i dati rispetto all'utilizzo di una feature categorica.

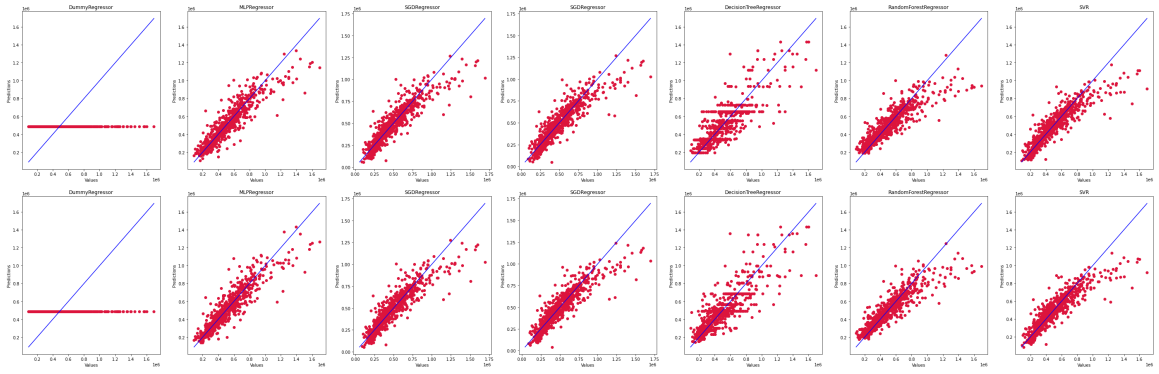

Figure 11: Grafici predizione/valore attuale per ogni modello, sopra allenati con il dataset originale, sotto con quello esteso. Si nota anche attraverso il grafico il miglioramento nel decision tree.

4.1.1 Clustering

Possiamo ulteriormente migliorare le performance dei modelli allenati con il dataset esteso attraverso l'utilizzo del clustering come spiegato nella sezione 3.2.

	Cluster 1 (304/751)		**Cluster 2** (206/751)		**Cluster 3** (241/751)		
	RMSE	MAE	RMSE	MAE	RMSE	MAE	Tot MAE
Rete neurale	120621.89	84109.27	54167.91	36626.51	107014.35	70194.79	**66619.47**
Lasso	122742.48	85282.69	53754.36	37012.46	110568.84	72466.94	**67929.47**
Ridge	126733.21	86639.71	54087.48	36824.57	111961.88	73232.25	**68672.84**
Decision tree	175090.90	125769.12	71349.57	48866.83	139656.11	86026.76	**91921.21**
Random forest	133296.10	92910.04	57140.54	40125.23	130805.02	76968.65	**73315.43**
SVM	120161.80	82193.35	55079.35	36726.16	117846.77	75034.62	**67424.37**

Table 2: Risultati delle predizioni combinando apprendimento supervisionato e K-means.

Tra parentesi accanto al numero del cluster sono indicati il numero di esempi classificati come appartenenti al cluster sul totale degli esempi di test. La colonna "Tot MAE" non è altro che la media tra tutti i cluster calcolata come:

$$\frac{\sum_{i=1}^{k} MAE_i \cdot |Cluster_i|}{\sum_{i=1}^{3} |Cluster_i|} \tag{3}$$

In questo caso riscontriamo qualche piccolo miglioramento negli stimatori **Lasso**, **Ridge** e **SVM**. Inoltre si può notare come alcuni stimatori funzionano meglio di altri su specifici cluster, si potrebbe quindi pensare di combinare modelli differenti su cluster diversi per massimizzare le performance (vedere la sezione 3.2).

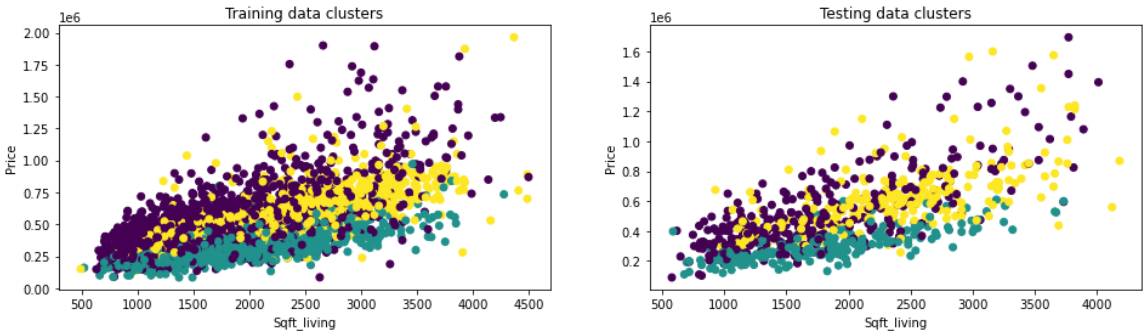

Figure 12: I tre cluster distribuiti sul training e testing set.

4.2 Train e test con città differenti

Utilizzando come funzione di splitting **train_test_diffcities_split()** simuliamo lo scenario nel quale durante il testing si incontrano case in città non presenti nel set di training (la dimensione dei set è sempre 80% training, 20% test). I risultati delle predizioni sono i seguenti.

In questo scenario il *mean average error* aumenta, quindi la qualità delle predizioni diminuisce sostanzialmente rispetto ai risultati precedenti. In questo caso però i modelli allenati con il dataset esteso funzionano notevolmente meglio rispetto a quelli allenati con il dataset originale.

	Original data			Extended data			
	R2	RMSE	MAE	R2	RMSE	MAE	decr. MAE
Dummy	0	241153.04	189087.16	0	241153.04	189087.16	**0**%
Rete neurale	0.117	226651.86	182441.63	0.392	188103.04	141451.75	**28.978**%
Lasso	0.477	174465.26	137111.43	0.6	152593.23	109693.36	**24.995**%
Ridge	0.487	172658.18	134693.9	0.6	152514.45	109608.88	**22.886**%
Decision tree	-0.211	265360.75	197765.79	0.542	163191.47	108231.19	**82.725**%
Random forest	0.368	191684.83	149244.86	0.63	146678.88	104214.31	**43.21**%
SVM	0.439	180650.8	136976.35	0.599	152661.49	106088.53	**29.115**%

4.2.1 Clustering

Anche in questo caso possiamo combinare il clustering con questi stimatori per migliorare la qualità delle predizioni.

	Cluster 1 (95/671)		**Cluster 2** (354/671)		**Cluster 3** (222/671)		
	RMSE	MAE	RMSE	MAE	RMSE	MAE	Tot MAE
Rete neurale	329103.37	265676.82	138318.85	106167.82	64009.57	45005.89	**108515.67**
Lasso	246069.36	209740.80	147408.77	100309.35	65457.82	46995.73	**98163.84**
Ridge	241350.01	205236.32	147258.72	100224.03	65334.73	46834.07	**97427.60**
Decision tree	279222.81	195405.96	177526.54	125350.35	83347.69	57663.32	**112874.59**
Random forest	231919.25	176273.30	156945.97	107479.84	68201.17	44938.32	**96527.77**
SVM	162850.36	123706.76	141990.15	99907.89	64456.98	42933.31	**84427.32**

Table 3: Risultati delle predizioni combinando apprendimento supervisionato e K-means.

Quasi tutti i modelli traggono benefici dall'uso del clustering, in particolare la **SVM** che passa da un *mean average error* di circa 106000 ad uno di 84400.

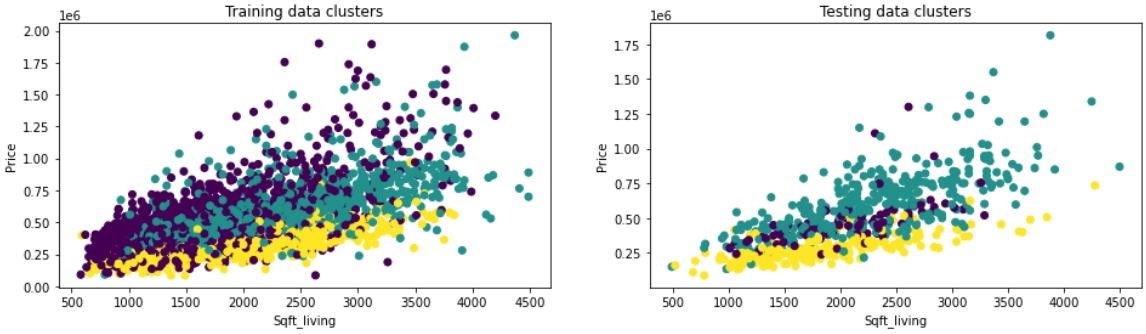

Figure 13: I tre cluster distribuiti sul training e testing set.