Data Mining II
Advanced Topics and Applications

Anno Accademico 2019/2020

Alessandro Andriotto, mat n. 600975

Alessandro Bonini, mat n. 604482

Pasquale Gorrasi, mat n. 597817

Indice

Il nostro progetto si propone di effettuare previsioni riguardanti il giorno della settimana in base alla composizione dell'aria rilevata tramite un dispositivo multisensore situato in un'area altamente inquinata nei dintorni di una città Italiana. (i dati sono stati raccolti dalla UCI Machine Learning Repository e le rilevazioni sono state effettuate da Marzo 2004 a Febbraio 2005).

Data Understanding

Il dataset *AirQualityUCI.csv* è composto di 9358 record.
Il dataset è composto da 15 attributi, quali: *date, time, CO(GT), PT08.S1(CO), NMHC(GT),C6H6(GT), PT08.S2(NMHC), NOx(GT), PT08.S3(NOx), NO2(GT), PT08.S4(NO2), PT08.S5(O3), T, RH, AH.*

Attributo	**Tipo**	**Tipologia**
date	*data*	data dell'osservazione
time	*data*	Orario dell'osservazione
CO(GT)	*continuo*	Concentrazione totale media oraria di CO
PT08.S1(CO)	*continuo*	Concentrazione totale media oraria di ossido di stagno
NMHC(GT)*	*continuo*	Concentrazione totale media oraria di idrocarburi non metanici
C6H6(GT)	*continuo*	Concentrazione totale media oraria di benzene
PT08.S2(NMHC)	*continuo*	Concentrazione totale media oraria di titanio
NOx(GT)	*continuo*	Concentrazione media oraria di ossidi di azoto in parti per miliardo
PT08.S3(NOx)	*continuo*	Concentrazione media oraria di ossido di tungsteno su base NOx
NO2(GT)	*continuo*	Concentrazione media oraria totale di diossido di azoto
PT08.S4(NO2)	*continuo*	Concentrazione media oraria di ossido di tungsteno su base NO2
PT08.S5(O3)	*continuo*	Concentrazione totale media oraria di ossido di indio
T	*continuo*	Temperatura

RH	*continuo*	Umidità espressa in valore percentuale
AH*	*continuo*	Umidità espressa in valore assoluto

Tabella 1: dati disponibili, tipo di dato e significato

Sulla base dei dati disponibili abbiamo creato una nuova variabile target, weekend, al fine di classificare i record.

weekend	*booleana*	indica se il giorno è parte del weekend (sabato, domenica) o meno.

Data l'alta correlazione degli attributi *PT08.S1, PT08.S2, PT08.S4, PT08.S5*, abbiamo deciso di unire i record sotto un unico attributo *PT08_feat* per ridurre la dimensionalità del dataset.

PT08_feat	*continuo*	media dei valori dei quattro attributi PT08 sopra citati

*E' stato eliminato l'attributo *NMHC(GT)* perchè composto prevalentemente da missing values.
*E' stato eliminato *AH* perché ritenuto ridondante rispetto ad *RH*.

I missing values contrassegnati con - 200 sono stati sostituiti con la media del rispettivo attributo mentre i valori nulli contrassegnati da Nan sono stati eliminati dato che sono riferiti a rilevazioni composte interamente da valori nulli.
Prima di effettuare la classificazione osserviamo la distribuzione della variabile target weekend.

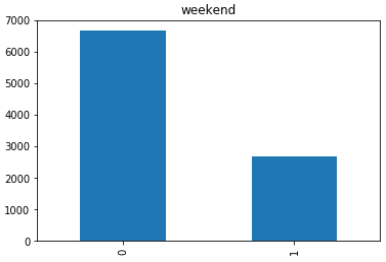


Figura 1: distribuzione della variabile decisionale

Task 1 - Basic Classifiers and Evaluation

Ogni algoritmo di classificazione è stato applicato seguendo lo stessa preparazione del dataset e lo stesso partizionamento dei dati. Abbiamo utilizzato l'80% come training per il modello e il restante 20% come test. Per gli algoritmi della task 1 è stato utilizzata la funzione *GridSearch* per ricercare i parametri migliori, fatta eccezione per il Naive Bayes che non richiede parametri. Infine i risultati vengono valutati utilizzando la confusion matrix, la roc curve/auc e lift chart

K-Nearest Neighbor

Il primo algoritmo di classificazione da noi usato è k-nearest neighbors (knn). Il parametro *n_neighbors* è stato testato su un range da 1 a 100, mentre il parametro weights è stato testato con i valori *uniform* e *distance*.
I migliori parametri ottenuti dalla ricerca ci hanno portato a selezionare un numero di vicini pari a 9, mentre "distance" si è rivelato il valore migliore per il parametro weights. Per ottenere una stima più precisa della bontà dell'algoritmo è stata applicata una cross-validation su 5 folds.

Algoritmo	Accuracy	Precision	Recall	F1 score	C. Matrix
KNN	0.738	0.642	0.492	0.703	[[1191, 1143], [264, 274]]

Tabella 2: prestazioni algoritmo KNN

Il basso score del Recall mostra come l'algoritmo sia piuttosto preciso per quanto riguarda l'individuazione dei giorni non weekend, ma ha difficoltà nell'individuare i giorni weekend.
La ROC curve mostra una buona performance del classificatore, con un AUC pari a 0.812.

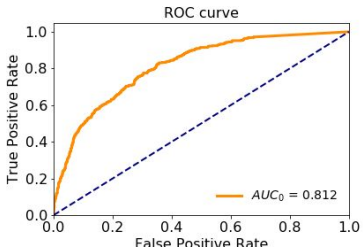


Figura 2a: roc curve KNN

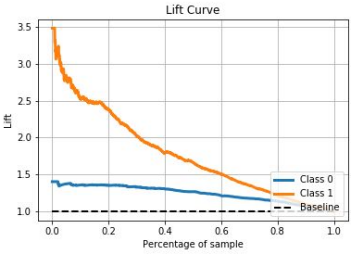


Figura 2b. lift chart KNN

La lift curve mostra come il modello riesca a classificare la classe 1 con un buon lift fino al 40% circa del dataset totale, i risultati sono invece meno buoni per quanto riguarda la classificazione della classe 0 che non si scosta dalla baseline neanche per una popolazione molto ridotta.

Naive - Bayes classifier

Abbiamo effettuato la classificazione con la tecnica del Naive-Bayes utilizzando l'algoritmo *GaussianNB*, approssimando quindi le variabili ad una distribuzione gaussiana.
I risultati ottenuti sono:

Algoritmo	Accuracy	Precision	Recall	F1 score	C. Matrix
Naive Bayes	0.576	0.373	0.701	0.563	[[702, 632], [161, 377]]

Tabella 3: prestazioni algoritmo Naive-Bayes

Questo classificatore mostra evidenti problemi nel classificare i giorni non weekend ma rispetto al knn ha un migliore valore di recall, perciò ha un miglior rapporto true positive/all positive.
Le prestazioni non ottimali sono confermate dalla ROC Curve che ci mostra come i risultati del classificatore non si discostano molto da quelli di un classificatore randomico con un AUC di 0.658.

Logistic Regression

In questa fase abbiamo testato l'algoritmo *LogisticRegression* con diversi valori del parametro *C* e abbiamo notato che per valori inferiori o uguali a 0.01 la classificazione assegna solo *non weekend*, quindi abbiamo un'accuracy di 0.712, ma sia recall che precision sono uguali a 0 dato che non viene

mai classificato *weekend*. Per valori superiori a 0.01 invece abbiamo una diminuzione dell'accuracy a scapito di precision e recall che diventano rispettivamente 0.428 e 0.011.
Riportiamo i risultati migliori ottenuti con il parametro *C* impostato a 10:

Algoritmo	Accuracy	Precision	Recall	F1 score	C. Matrix
Logistic Regression	0.711	0.428	0.011	0.43	[[1326, 8], [532, 6]]

Tabella 4: prestazioni algoritmo Logistic Regression

La confusion matrix mostra come la logistic regression classifichi quasi tutti i record come 'non weekend', generando un numero elevato di false negative.
I grafici di ROC Curve e Lift chart mostrano anche in questo caso come la performance del classificatore sia molto vicina a quella del classificatore random, risultando quindi scarsamente efficiente. Rispettivamente la ROC Curve evidenzia un AUC di 0.572 e la Lift chart mostra come le previsioni delle classi target siano molto vicine alla baseline anche per valori bassi di popolazione presa in considerazione.

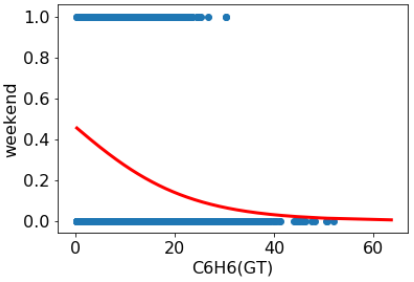


Figura 3: grafico logit C6H6(GT)

Decision Tree

Dopo aver implementato i precedenti algoritmi, abbiamo effettuato il decision tree.
Per il tuning dei parametri ci siamo avvalsi della funzione *GridSearch* con una cross validation basata su 5 folds, testando diverse misure d'errore, quali gini ed entropia, diversi numeri di record nelle foglie dell'albero (da 1 a 10) e diversi livelli minimi per lo split (da 2 a 10) .
L'entropia si è rivelata la metrica più adatta per misurare l'errore, il numero di record ottimale per foglia è dieci (non sono state riscontrate sostanziali differenze al variare di questo parametro) e il livello minimo per lo split uguale a 3. L'accuracy media ottenuta dal classificatore così impostato è di 0.726 con deviazione standard 0.009
Il modello ottimizzato con i parametri trovati attraverso la *gridsearch* ottiene i seguenti risultati:

Algoritmo	Accuracy	Precision	Recall	F1 score	C. Matrix
Decision Tree	0.726	0.495	0.461	0.638	[[1081, 253], [290, 248]]

Tabella 5: prestazioni algoritmo Decision Tree

Dopo aver ottenuto questi risultati, abbiamo visualizzato il decision tree e l'importanza di ogni feature nel modello.

La feature C6H6(GT), come evidenziato anche dal grafico sottostante determina per gran parte la classificazione del decision tree, con i primi due split che dipendono interamente dai suoi valori.

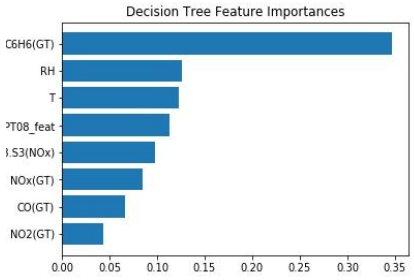


Figura 4a: decision tree feature importance

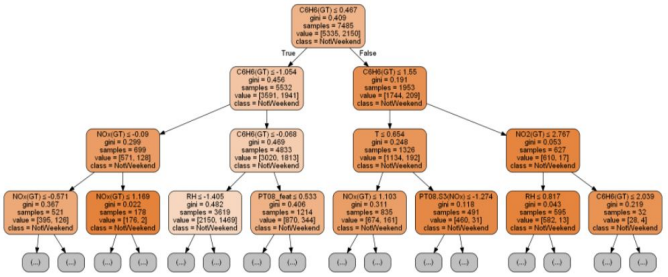


Figura 4b: decision tree

La ROC Curve evidenzia un AUC di 0.703.

Dimensionality reduction

Abbiamo ora applicato diversi metodi per la riduzione della dimensionalità del dataset, in particolar modo Variance Threshold (VT), Univariate Feature Selection (UFS), Recursive Features Elimination (RFE) e Principal Component Analysis (PCA). Dopo aver ridotto la dimensionalità del dataset sono stati applicati algoritmi di classificazione quali decision tree e knn, valutandone non solo le metriche ma anche le relative confusion matrix.

Variance Threshold

Il primo metodo di riduzione della dimensionalità da noi testato è il variance threshold, selezionando di volta in volta un valore di *variance* tale per cui un attributo veniva rimosso ad ogni iterazione. Sintetizziamo in formato tabellare/grafico risultati ottenuti con i diversi algoritmi

DECISION TREE					
N. Attributi	Accuracy	Precision	Recall	F1-Score	C. Matrix
7	0.711	0.496	0.190	0.547	[[1845 156] [653 154]]
6	0.715	0.519	0.114	0.507	[[1916 85] [715 92]]
5	0.717	0.55	0.101	0.5	[[1934 67] [725 82]]
4	0.714	0.592	0.019	0.435	[[1990 11] [791 16]]
3	0.715	0.625	0.024	0.44	[[1989 12] [787 20]]
2	0.715	0.567	0.047	0.459	[[1972 29] [769 38]]

1	0.715	0	0	0.415	[[1999 2] [807 0]]

Tabella 6: prestazioni decision tree al variare delle features

KNN					
N. Attributi	Accuracy	Precision	Recall	F1-Score	C. Matrix
7	0.761	0.659	0.353	0.653	[[1854 147] [522 285]]
6	0.758	0.651	0.340	0.646	[[1854 147] [532 275]]
5	0.748	0.621	0.320	0.631	[[1843 158] [548 259]]
4	0.737	0.596	0.267	0.601	[[1855 146] [591 216]]
3	0.728	0.583	0.194	0.561	[[1889 112] [650 157]]
2	0.722	0.658	0.069	0.48	[[1972 29] [751 56]]
1	0.711	0	0	0.415	[[1997 4] [807 0]]

Tabella 7: prestazioni knn al variare delle features

Come si può notare le prestazioni dei due algoritmi convergono al diminuire del numero di features. Tuttavia, all'aumentare delle features il knn fornisce prestazioni generalmente migliori rispetto al decision tree.

Univariate feature selection

E' stato poi testato il metodo dell'*univariate feature selection*, selezionando di volta in volta un valore di k diverso, partendo da una feature per arrivare poi ad avere il dataset completo in tutte le sue dimensioni. Ad ogni iterazione è stato applicato sia il DT che il KNN ottimizzando i parametri tramite GridSearch e valutandone le performance tramite le tre metriche e la confusion matrix. I risultati ottenuti non si discostano significativamente dai precedenti mostrati in tabella, ragion per cui non esponiamo i risultati ottenuti ad ogni passaggio.

Recursive feature elimination

E' stato testata la tecnica del recursive features elimination, passando come parametri "estimator" il DecisionTreeClassifier, "step" = 1, min_feature_to_select = 1, "cv" = 5, scoring = "accuracy", il

risultato è che l'algoritmo seleziona comunque tutte le features del dataset per effettuare la previsione, ottenendo risultati equivalenti a quelli della classificazione effettuata sull'intero dataset.

PCA

Abbiamo poi effettuato una riduzione della dimensionalità del dataset applicando la Principal Component Analysis. E' stato ridotto il dataset mantenendo due attributi, ovvero quelli che catturano la maggior variabilità dei dati. Questi sono risultati essere, in ordine, CO(GT) e C6H6(GT), come si può notare dal seguente grafico:

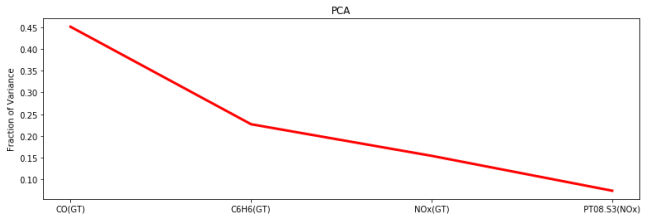


Figura 5: varianza features pca

Come si può notare questi due attributi tuttavia non catturano tutta la variabilità del dataset. E' stato visualizzato anche lo scatterplot dei due autovettori più importanti.

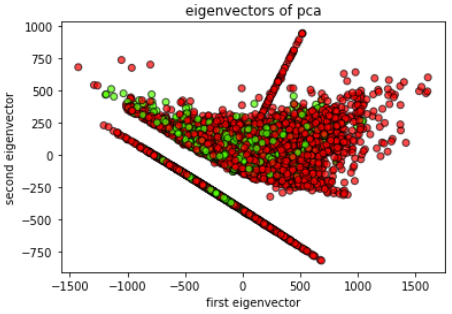


Figura 5: autovettori pca

A questo punto sono stati applicati gli algoritmi di classificazione del Decision Tree e del KNearestNeighbors ottimizzando i parametri come solitamente fatto e valutandoli come al solito. Ne risulta quanto segue:

Algoritmo	Accuracy	Precision	Recall	F1score	C. Matrix
Decision Tree	0.713	1	0.024	0.418	[[2001 0] [805 2]]
KNN	0.707	0.459	0.09	0.487	[[1915 86] [734 73]]

Tabella 8: prestazioni knn e decision tree dopo pca

Come si può notare non ci sono grandi scostamenti di prestazioni tra i due algoritmi, sia in termini di prestazioni, sia in termini di errori di classificazione. Quello che risulta interessante notare è come, nonostante ci siano due attributi soltanto, le prestazioni degli algoritmi di classificazione si avvicinano di molto a quelle degli algoritmi applicati all'intero dataset.La ROC Curve evidenzia un AUC di 0.644.

Dopo aver applicato la PCA abbiamo osservato i decision boundaries di vari algoritmi. Di seguito ne riportiamo le immagini:

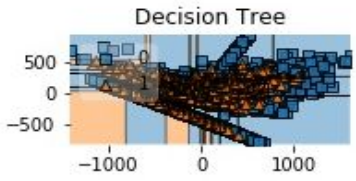


Fig. 6a: Decision Tree

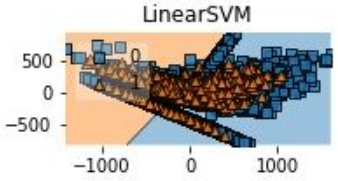


Fig 6b: Linear SVM

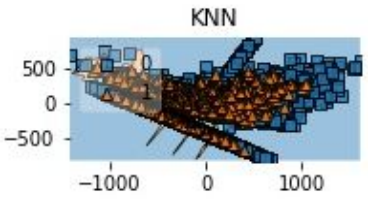


Fig. 6c KNN

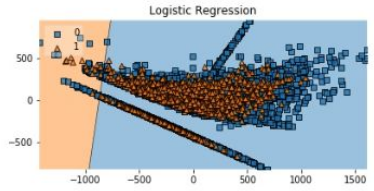


Fig. 6d: Logistic Regression

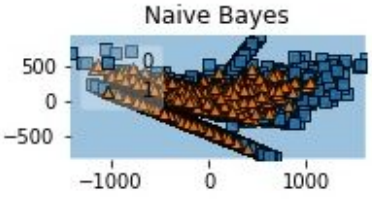


Fig. 6e: NaiveBayes

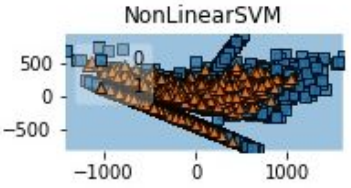


Fig. 6f: NonLinearSVM

Imbalanced dataset

Abbiamo creato un dataset fortemente sbilanciato, 96%-4%. All'inizio abbiamo rimosso record random aventi la classe 1, mantenendo pertanto il 96% di record appartenenti alla classe 0 e testando in prima istanza i soliti due algoritmi più prestanti tra quelli già provati.
Sbilanciando il dataset in tal modo si può notare come il decision tree predica correttamente tutti i record appartenenti alla classe 0, mentre non riesce a classificare correttamente i record appartenenti alla classe 1. L'accuracy ottenuta è infatti pari a 0.96. E' stato poi testato il knn ottimizzando i parametri tramite GridSearch, e le prestazioni sono risultate equivalenti.

Sulla base del dataset sbilanciato abbiamo poi performato l'Undersampling tramite due tecniche: *RandomUndersampling* (RUS) e *CondensedNearestNeighbours* (CNN), ripetendo la classificazione coi due algoritmi. Tramite il RUS abbiamo ottenuto un dataset composto di 209 record per ogni classe, mentre per il *CondensedNearestNeighbors* abbiamo ottenuto 741 record per la classe zero,mentre 209 per la classe 1.
Riportiamo di seguito le prestazioni ottenute, analizzandole nelle solite modalità.

Algoritmo	Accuracy	Precision	Recall	F1 score	Confusion Matrix
RUS DT	0.956	0	0	0.489	[[1313 689]

					[40 49]]
RUS KNN	0.589	0.061	0.606	0.422	[[1179 823] [35 54]]
CNN DT	0.589	0.06	0.606	0.422	[[1947 55] [86 3]]
CNN KNN	0.956	0	0	0.489	[[2001 1] [89 0]]

Tabella 9: knn e dt con RandomUnderSampling e CondensedNearestNeighbors

Come si può notare il *RandomUnderSampling* per il decision tree fornisce prestazioni leggermente migliori rispetto al knn, mentre il contrario si può affermare per il *CondensedNearestNeighbours*. Le prestazioni rimangono comunque insoddisfacenti, ma questo è probabilmente dovuto alla presenza di pochi record sui quali l'algoritmo possa imparare per effettuare una classificazione precisa delle istanze. Nel caso del CNN, avendo comunque più record della classe 0, si può notare come le classificazioni per questa classe siano migliori rispetto al caso del *RandomUnderSampler*. Rimangono insoddisfacenti le prestazioni nella classe 1.
Abbiamo poi performato l'Oversampling, seguendo le tecniche del *RandomOversampling* (ROS) e dello Smote (SM). Con entrambe le tecniche sono stati ottenuti training set composti di 4667 record per entrambe le classi, e su questi training sono stati allenati gli algoritmi di classificazione. Ne riportiamo le metriche e la confusion matrix.

Algoritmo	Accuracy	Precision	Recall	F1 score	Confusion Matrix
ROS DT	0.956	0	0	0.489	[[1636 366] [56 33]]
ROS KNN	0.919	0.065	0.067	0.512	[[1916 86] [83 6]]
SM DT	0.77	0.086	0.460	0.506	[[1551 451] [55 34]]
SM KNN	0.853	0.064	0.179	0.507	[[1768 234] [73 16]]

Tabella 10: knn e dt con RandomOverSampling e Smote

Le prestazioni degli algoritmi non si discostano molto dalla casistica precedente. Gli algoritmi commettono errori di classificazione, e faticano comunque a predire la classe 1.

Regressioni Lineari

Abbiamo performato tre modelli di regressioni lineari: lineare semplice, Ridge e Lasso.

I modelli di regressione sono stati applicati selezionando come attributi C6H6(GT), e *PT08_feat*. Riportiamo in formato tabellare i vari risultati, usando come metriche di valutazione il coefficiente di determinazione (R2), lo scarto medio quadratico (MSE) e lo scarto medio assoluto (MAE)

Regressione	R2	MSE	MAE	Formula
Lineare Semplice	0.878	7.343	1.952	C6H6(GT) = -18.95 + 0.025 * pt08_feat
Ridge	0.643	23819.123	123.116	C6H6(GT) = -267.06 + 4.21 * PT08_feat
Lasso	-1061.167	63805.939	186.090	C6H6(GT) = -18.936 + 0.025*pt08_feat :

Tabella 11: prestazioni regressioni

riportiamo graficamente il caso di regressione più performante, ovvero quello della regressione lineare semplice.

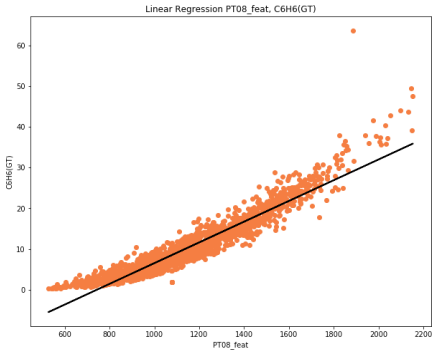


Figura 7: regressione lineare

Conclusioni

I risultati peggiori ottenuti in questa fase derivano dalla regressione logistica, la quale si comporta come un classificatore random. Per quanto riguarda i restanti tre possiamo dire che il *naivebayes* è l'unico che ha un valore di recall buono, ma risulta pessimo in tutte le altre metriche. Il decision tree, che vede C6H6(GT) come suo attributo più importante, ha una performance accettabile; infine il knn che appare superiore in tutte le metriche (anche se ha un punteggio di recall pari a 0.492) ed arriva a un valore AUC di 0.812. Come si può generalmente notare, la predizione della classe 0 non riscontra grossi problemi, mentre risulta difficile per gli algoritmi predire la classe 1.

Task 2 - Advanced Classifiers and Evaluation

Support Vector Machines (SVM)

Linear SVM

Il primo algoritmo che abbiamo testato in questa fase è il *linearSVM* che abbiamo valutato al variare del parametro C, i valori testati sono: 0.1, 1, 10, 100 ,1000. Il risultato di questa analisi però appare uguale per qualsiasi valore di C ed il motivo si può capire osservando sia la confusion matrix che la ROC curve, questo classificatore infatti non è in grado di fornire alcuna informazione in quanto predice solo 0, ossia 'non weekend'. Da questo possiamo dedurre che ci troviamo di fronte un problema non risolvibile linearmente.

SVC

Nell'analisi effettuata utilizzando il classificatore *NonLinearSVC* e la *gridsearch* abbiamo valutato i seguenti parametri: ('C': da 0.01 a 100 con intervalli di 10, 'gamma': 1,0.1,0.01,0.001 e 'kernel': 'rbf', 'poly', 'sigmoid', 'linear'). Il risultato migliore è stato ottenuto con i parametri: ('C': 90.01, 'gamma': 1, 'kernel': 'rbf'), con un risultato in accuracy pari a 0.736, che è superiore al classificatore precedente. Il punteggio in Precision è 0.733 mentre quello in Recall 0.122. Anche qui possiamo notare come il classificatore non riesca bene nel suo compito, riuscendo a classificare correttamente solo 66 dei 538 valori *weekend*. Risultato confermato dal punteggio f1-score macro che è pari a 0.53. Dato che il kernel trick si è rivelato inefficiente, deduciamo che questo metodo di classificazione non sia adatto alla struttura del nostro dataset.

Confusion Matrix
[[1310, 24] [472, 66]

Tabella 12: Confusion Matrix SVM

Neural Networks and Deep Learning

In questa fase abbiamo testato *MLPClassifier* e *keras.* Abbiamo iniziato la nostra analisi usando l'*MLPClassifier* con singolo hidden layer e anche in questo caso, non essendo il nostro problema linearmente risolvibile, i risultati non sono buoni e tendono a quelli di un classificatore random.
Successivamente abbiamo aumentato il numero degli hidden layers da uno a tre e testato manualmente i principali parametri dell'*MLPClassifier*:

1) Sono stati selezionati 3 hidden layers (128,64,32) poichè un incremento del numero degli hidden layers incrementa in maniera minima il risultato ma rende più complesso il modello.
2) Il valore ideale di alpha è 0.01 in quanto un numero minore genera un incremento insignificante mentre per valori maggiori come 1 abbiamo un peggioramento delle prestazioni, fino ad arrivare ad un risultato analogo a quello dei classificatori lineari in cui abbiamo un valore di recall uguale a 0 quando impostiamo alpha uguale a 10
3) Per quanto riguarda il *learning_rate* abbiamo notato che il risultato rimane uguale per i tre valori: 'adaptive'. 'invscaling', 'constant'.
4) Abbiamo provato diverse activation function tra cui: 'relu', 'tanh', 'logistic', 'identity' e abbiamo raggiunto i risultati migliori con relu e tanh.

5) I risultati ottenuti senza early_stopping sono leggermente migliori di quelli ottenuti utilizzando l'early_stopping.

Il modello ottimizzato quindi presenta i seguenti parametri: hidden_layer_sizes = (128, 64, 32), alpha = 0.01, learning_rate = 'adaptive', activation = 'relu', early_stopping = False. I risultati sono:

Algoritmo	Accuracy	Precision	Recall	F1 score	AUC
MLPClassifier	0.794	0.661	0.589	0.742	0.839

Tabella 13: prestazioni MLPClassifier

Il secondo passo è stato quello di creare diversi modelli di deep neural network usando la libreria keras. Per ogni modello sono stati lasciati invariati optimizer = 'adam' e loss_function = 'binary_crossentropy' essendo un problema di classificazione binaria.
Attraverso i vari modelli testati siamo arrivati alla conclusione che la funzione di attivazione 'sigmoid' sia la migliore da utilizzare per l'output layer (perchè siamo di fronte a un problema di classificazione binaria), mentre 'relu' si presta bene per gli hidden_layers. Abbiamo scelto una struttura (128,64,1) e analizzato le prestazioni di 8 modelli variando le funzioni di attivazione dei due hidden layers e dell'output layer. La prestazione migliore è data dal modello che prevede 2 hidden layers rispettivamente da 128 neuroni e 64 neuroni aventi 'relu' come funzione di attivazione e un output layer composto da 1 neurone con 'sigmoid' come funzione di attivazione. Con questo modello otteniamo un'accuracy di 0.782, la loss_function è pari a 0.468, precision 0.665 e recall 0.487. Applicando l'early stopping con 'Patience' di 5 iterazioni invece, abbiamo un'accuracy di 0.761 e loss function 0.494. Confrontando lo stesso modello con differenti valori di batch_size è apparso conveniente selezionare un numero non elevato, infatti un valore di 10 è preferibile ad uno di 50 anche se computazionalmente più costoso; per quanto riguarda il numero di epochs invece, il numero ideale è 1 poichè attraverso un'analisi grafica della cross entropy sia del training che della validation è stato notato che fino ad 1 la loss function diminuisce in entrambi i dataset, mentre per valori superiori la loss function del validation dataset inizia ad aumentare dopo l'iniziale discesa e la loss function del training continua a diminuire in maniera graduale sempre di più. Ciò indica che dopo la prima epoch il modello inizia ad andare in overfitting. Infine è stato creato un modello con 4 hidden layers aventi 'relu' come activation function ed utilizzando la regolarizzazione ridge (l2). Il risultato ottenuto è di 0.721 in accuracy e 0.591 per la loss function.

Ensemble Learning

In questa fase abbiamo analizzato i risultati ottenuti tramite tre tipi di classificazione: random forest, bagging e boosting.

Random Forest

Per prima cosa abbiamo analizzato la feature importance del randomforest attraverso una rappresentazione grafica:

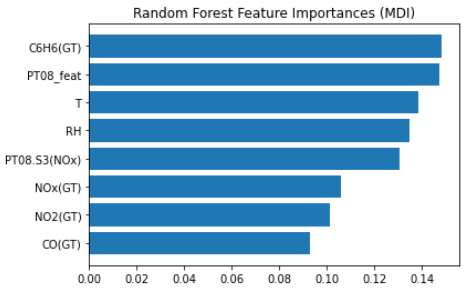


Figura 8: Random Forest feature importance

Da cui possiamo notare che l'importanza degli attributi è più bilanciata, seppure il più importante rimanga C6H6(GT).
Successivamente parametri ottimali attraverso l'analisi di diverse randomsearch aventi come parametri fissi: 'max_depth': da 2 a 20, 'min_samples_split': [2, 5, 10, 20, 30, 50, 100], 'min_samples_leaf': [1, 5, 10, 20, 30, 50, 100], 'max_features' : ['auto', 'sqrt', 'log2'], mentre abbiamo variato lo scoring con le seguenti metriche: roc_auc,accuracy,recall e f1_macro. Tra i vari modelli quello che abbiamo ritenuto migliore è quello con parametri: max_depth = 19, max_features = 'auto', min_samples_split = 10, min_sample_leaf = 1 e scoring = 'f1_macro'. I risultati ottenuti da questo modello sono:

Algoritmo	Accuracy	Precision	Recall	F1 score	AUC
Random Forest	0.769	0.687	0.361	0.663	0.818

Tabella 14: prestazioni RandomForest

Bagging

Utilizzando la tecnica del bagging abbiamo testato i classificatori che si sono comportati meglio negli step precedenti, abbiamo analizzato quindi l'algoritmo baggingclassifier con i seguenti classificatori base: knn, naive bayes, decision tree, tutti impostati con i loro parametri ottimali trovati con le ricerche gridsearch precedenti. Il numero di iterazioni e di classificatori base è stato impostato a 100 dato che ci è sembrato un buon compromesso tra complessità e risultati.

Algoritmo	Accuracy	Precision	Recall	F1 score	AUC
NaiveBayes	0.577	0.374	0.702	0.564	0.659
KNN	0.731	0.544	0.373	0.63	0.737
Decision Tree	0.776	0.659	0.457	0.696	0.815

Tabella 15: prestazioni bagging con KNN, DT, NB

Boosting

L'ultima metodologia riguardante gli ensemble che abbiamo testato è quella che si avvale dell'uso dell'algoritmo *adaboostclassifier*, ovvero il boosting. In questa fase abbiamo provato gli stessi classificatori usati in precedenza per il bagging ad eccezione del knn. I classificatori che hanno ottenuto un incremento dovuto alla metodologia del boosting sono: il GaussianNB che nonostante continui ad avere pessimi valori quasi in tutti gli score ma in recall arriva fino a 0.829 e il random forest che incrementa il suo punteggio in AUC fino a 0.828. Comportamento curioso del DT che peggiora di molto la prestazione.

Conclusioni

Riassumendo possiamo dire che nel support vector è preferibile il modello non lineare a quello lineare pur avendo pessimi risultati dato che quest'ultimo produce risultati uguali a quelli di un classificatore random. Tuttavia a causa dei cattivi risultati in entrambi i casi possiamo dire che è una metodologia che non si adatta al nostro tipo di problema sospettiamo a causa di una mancanza di dati disponibili nel training set. Nell'utilizzo dei neural networks abbiamo avuto risultati relativamente buoni, in particolar modo con i deep neural networks. Abbiamo inoltre notato che l'activation function ideale per gli hidden layers è 'relu' mentre per l'output layer 'sigmoid'.
Infine sperimentando vari modelli con l'ensemble abbiamo ritenuto che un buon numero di iterazioni e di classificatori base fosse 100, trovando un compromesso tra complessità e bontà dei risultati.

Task 3 - Time Series

Durante lo svolgimento di questa task per prima cosa abbiamo sostituito le colonne 'Date' e 'Time' con l'attributo 'Datetime' e abbiamo impostato quest'ultimo come indice. Le time series che abbiamo analizzato appartengono alle features più importanti secondo il random forest analizzato in precedenza, ossia: PT08_feat, RH, C6H6(GT) e T; gli split temporali testati invece sono: giornaliero, settimanale, mensile e stagionale. Abbiamo preferito quello settimanale nella nostra analisi dato che lo split giornaliero avrebbe creato time series troppo brevi, mentre con split mensile o stagionale avremmo ottenuto troppe poche time series per trainare i classificatori.

Time series similarities

Per confrontare lo stesso attributo in settimane diverse è stato necessario resettare l'indice altrimenti otteniamo grafici con le time series una davanti all'altra. I valori vanno dalla mezzanotte di lunedì alle 23.59 della domenica successiva.
Abbiamo condotto l'analisi confrontando la prima settimana di ogni stagione. Di seguito presentiamo l'andamento degli attributi Abbiamo utilizzando la prima settimana di primavera per confrontare. In ultimo è presente il grafico con la sovrapposizione degli attributi, per fare ciò è stato necessario normalizzare i valori dato che l'attributo pt08_feat andava a prevalere sugli altri finendo per rendere non leggibile il grafico.

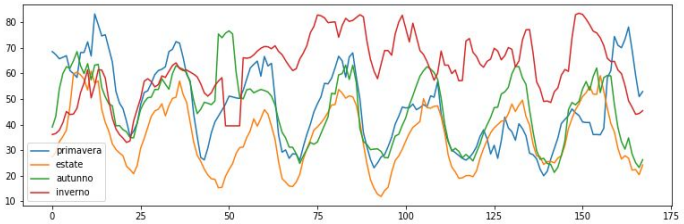


Figura 10: time series RH

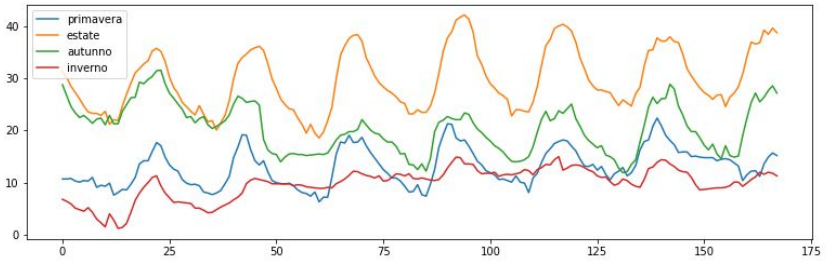


Figura 11: time series Temperature

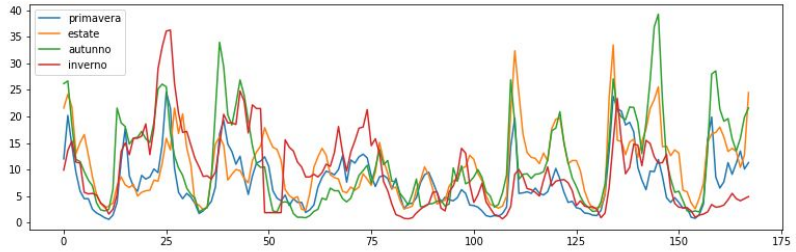


Figura 12: time series C6H6(GT)

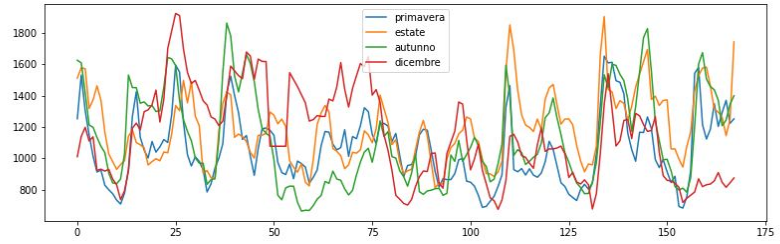


Figura 13: time series PT08_feat

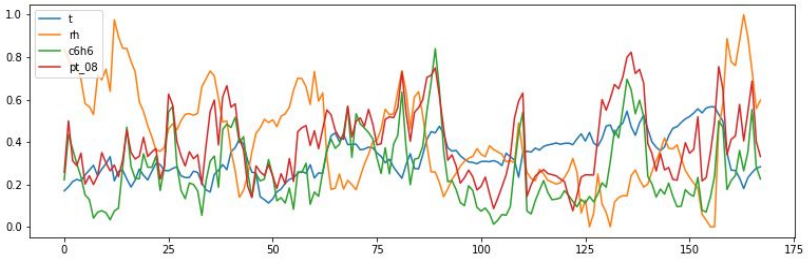


Figura 14: varie time series sovrapposte

Dai grafici possiamo notare che per quanto riguarda la temperatura abbiamo una seasonality per tutte le stagioni che si può attribuire al variare del giorno e della notte. l'amplitude della seasonality è maggiore in estate quando di giorno fa molto caldo e di notte le temperature scendono (siamo a milano). Per quanto riguarda RH, ossia l'umidità possiamo notare che d'inverno abbiamo un'alta percentuale di umidità per quasi tutta la settimana selezionata mentre per le altre stagioni ci troviamo di fronte a una seasonality attribuibile al passaggio da giorno a notte. Infine dall'ultimo grafico, ossia quello che mette a confronto la prima settimana di primavera tra i vari attributi possiamo notare che gli attributi c6h6 e pt08_feat hanno un andamento simile.

Clustering

In questa fase abbiamo utilizzato tecniche di clustering basate sulla forma e sulle caratteristiche delle time series usando rispettivamente l'algoritmo *TimeSeriesKMeans* ed il classico *KMeans*. Gli attributi analizzati sono *T* e *RH* ed in entrambi i casi abbiamo misurato l'inerzia attraverso la distanza euclidea, la dynamic time warping. Il numero di cluster è stato testato da 2 a 9.

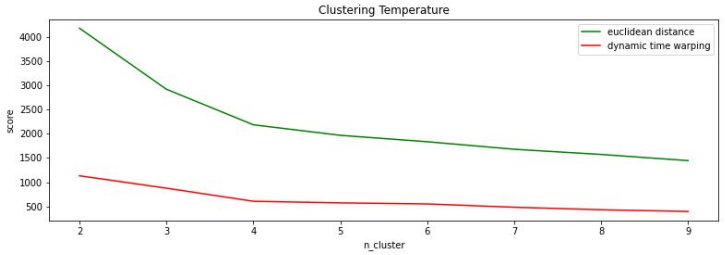


Figura 15: clustering temperature

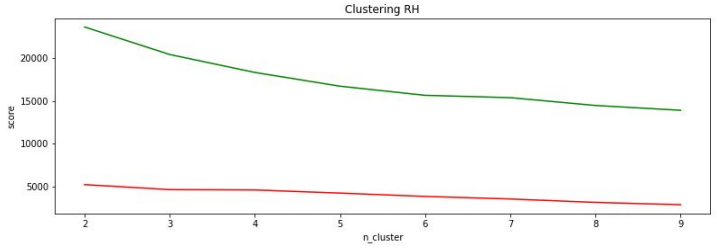


Figura 16: clustering rh

Utilizzando il knee method possiamo dire che il numero ideale di cluster per il primo attributo è 4, invece non abbiamo abbastanza informazioni utili per definire un numero ideale di cluster per il secondo attributo

La seconda analisi condotta usando il clustering è stata fatta utilizzando il metodo feature-based, ossia prendendo come valori le caratteristiche della lista di time series.

In questo caso testando il numero di cluster da 2 a 10 abbiamo ottenuto i seguenti risultati di inertia:

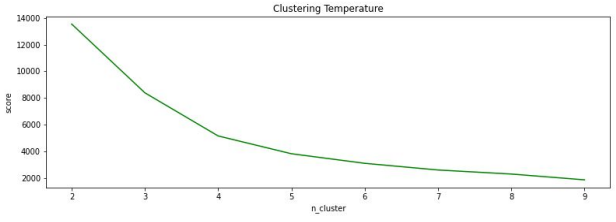


Figura 17: clustering temperature

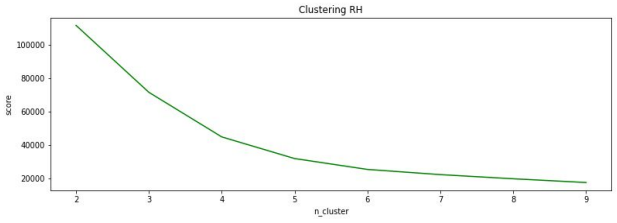


Figura 18: clustering rh

Dai grafici sovrastanti possiamo notare come il numero ideale *K* da selezionare per il *KMeans* sia tra 4 e 5.
I migliori risultati ottenuti dal clustering dell'attributo temperature possono essere giustificati da una maggiore separazione tra le varie settimane a seconda delle stagioni, facilitando così un buon clustering. Inoltre, come ci aspettavamo, notiamo chiaramente dei risultati migliori usando il dynamic time warping rispetto alla distanza euclidea che non riesce a cogliere la differente velocità di due time series.

Shapelets

Ci siamo avvalsi di un training set composto da 42 time series e un test set composto da 11 time series. La variabile target è stata impostata in modo da dividere le stagioni fredde da quelle calde, quindi abbiamo dato come label alle varie time series il valore 0 per le stagioni calde: estate e primavera, mentre 1 per le stagioni fredde: inverno e autunno. I risultati migliori sono stati ottenuti con l'attributo RH di cui riportiamo i grafici.

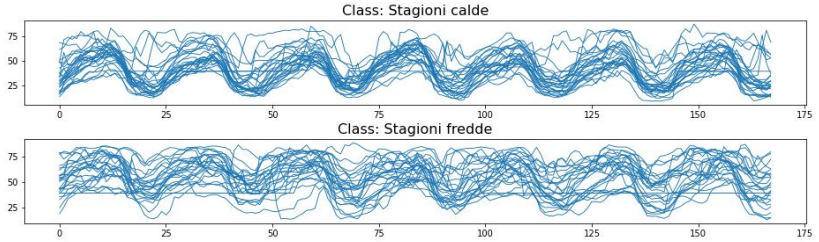


Figura 19: time series divise per class label

La dimensione ideale delle shapelets trovata grazie all'algoritmo grabocka_params_to_shapalet_dict è 3 ed a seguito della classificazione fatta utilizzando l'algoritmo shapeletModel con l'ottimizzatore impostato 'sgd' otteniamo i seguenti risultati:

Algoritmo	Accuracy	Precision	Recall	F1 macro
ShapeletModel	0.636	0.571	0.8	0.633

Tabella 16: prestazioni shapelet model

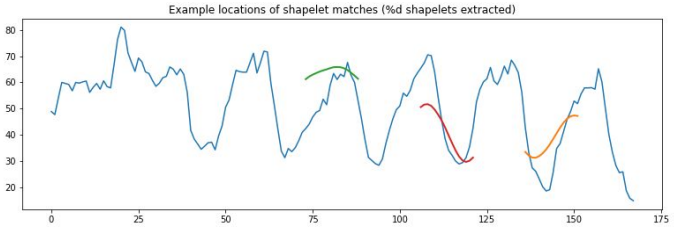


Figura 20: shapelets

Abbiamo approfondito l'analisi delle shapelets dell'attributo RH usando un secondo metodo di shapelet discovery ossia *shapeletTransform*.

Abbiamo selezionato il numero di shapelets pari a 3 e per la window size abbiamo variato da 2 a 10, in tutti i casi però purtroppo i risultati ottenuti secondo lo score utilizzato sono intorno allo 0.5. Ciò ci indica che queste shapelets non sono molto utili per una buona classificazione finale.
Qui sotto riportiamo le tre shapelet individuate per window size uguale a 8.

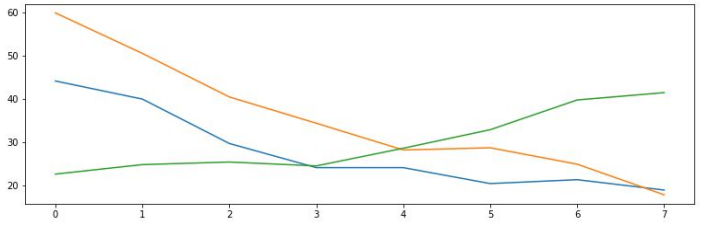


Figura 21: shapalets

I cattivi risultati potrebbero essere attribuiti al fatto che abbiamo solo 11 ts nel test set, ma splittare per intervalli più piccoli di settimane avrebbe creato ts troppo brevi per essere analizzate con questo metodo

Motif

L'attributo usato per il motif discovery è 'RH' nel periodo dal 2004-06-21 al 2004-09-20. Per il proposito di questa task, siamo partiti rappresentando graficamente la time series nella sua totalità.
Abbiamo notato una notevole varianza e per questo motivo abbiamo deciso di applicare una smoothing transformation per ridurre la variabilità della time series. A questo punto, abbiamo testato diversi window size per cercare di avere una matrix profile quanto più pulita possibile. Il window size ottimale trovato è pari a 168. La matrix profile risultante è la seguente:

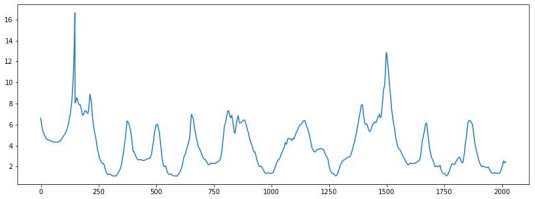


Figura 22: matrixprofile

Osservando questa matrixprofile possiamo notare come la time series estiva presenti due anomalie principali e diversi motif, per la precisione 4, trovati impostando a 6 il parametro max_motif dell'algoritmo motifs..

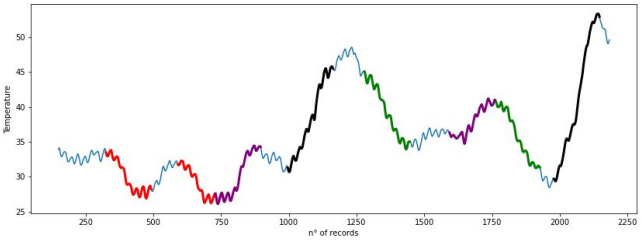


Figura 23: motif evidenziate

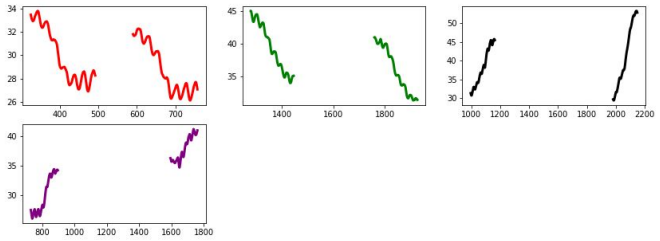


Figura 24: motif isolate

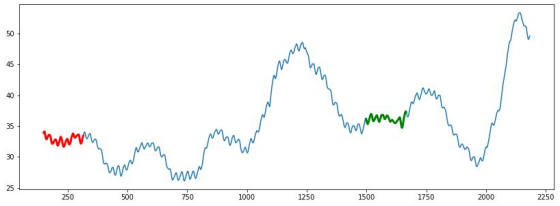


Figura 25: anomalies

FORECASTING

In questa task abbiamo selezionato l'attributo 'Temperature' per il periodo che va dal 2004-06-10 al 2004-07-01. La time series originale si presenta così:

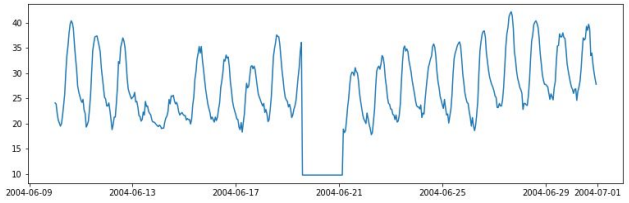


Figura 26: Time series temperature

Dal grafico possiamo notare una seasonality all'incirca di 24 valori ossia le ore giornaliere, per quanto riguarda il trend possiamo dire che è leggermente crescente. Analizziamo la TS con il test Dickey-Fuller per verificare la stazionarietà ed otteniamo i seguenti risultati:
Test Statistic = -1.718
Critical Value (1%) = -3.443
Critical Value (5%) = -2.867
Critical Value (10%) = -2.569

Da questi risultati possiamo notare che la ts non è stazionaria dato che l'ipotesi nulla non è respinta (test statistic ha un valore inferiore ai critic values) perciò abbiamo effettuato una trasformazione logaritmica della ts ed abbiamo eliminato il trend sottraendo a ciascun valore la media mobile a 12 periodi, ovvero il valore medio della temperatura nelle 12 ore precedenti. La ts così generata è stazionaria, infatti i risultati ottenuti dal test Dickey-Fuller sono:
Test Statistic = -9.631640e+00;
Critical Value (1%) = -3.444461e+00
Critical Value (5%) = -2.867762e+00
Critical Value (10%) = -2.570084e+00

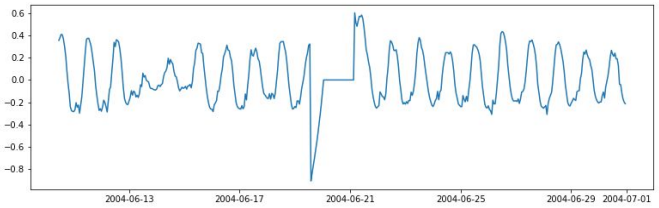


Figura 27: time series stazionaria

Di seguito inseriamo i grafici relativi all'autocorrelazione:

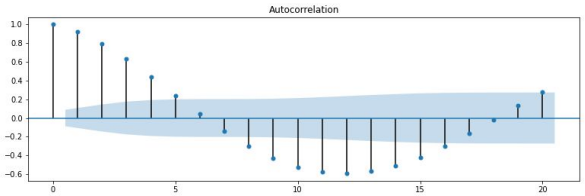


Figura 28: ACF

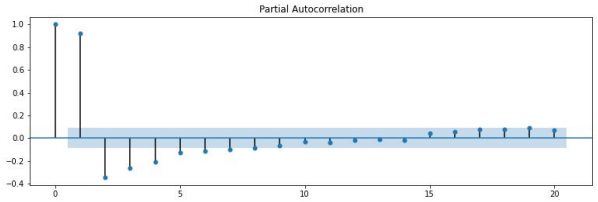


Figura 29: autocorrelazione parziale

Trovandoci di fronte ad un ACF sinusoidale e con due picchi significativi a lag 2 e non oltre nel PACF, possiamo dire che i parametri ottimali dell'ARIMA dovrebbero essere (2,d,0), ossia un modello che si avvale del'*autoregressive mode*l ma non del *moving average model.*
A questo punto siamo passati al forecasting della ts in questione, per fare ciò abbiamo utilizzato 3 dei 5 metodi proposti durante il corso, ossia: Holt-Winter's seasonal method, ARIMA e SARIMAX. Abbiamo provato ma escluso subito il simple exponential smoothing e l'Holt's linear method in quanto la nostra time series presenta una delle seasonality e questi due metodi non riescono a catturarla.

Holt-Winter's seasonal method

Abbiamo impostato questo algoritmo con un seasonal periods = 24, per catturare il pattern seguito dalla temperatura nell'arco della giornata, seasonal = 'add e trend = 'add.

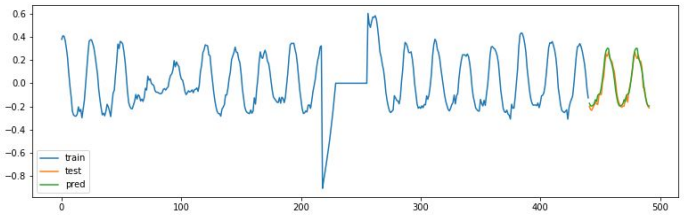


Figura 30: Holt-Winter's method

I risultati ottenuti sono:

MAE	0.031
RMSE	0.038
MAD	0.024
R2	0.951
MAPE	0.349
MAXAPE	2.837
TAPE	17.815

Tabella 17: prestazioni Holt-Winter's method

Questo modello come tutti quelli appartenenti alla famiglia dell'exponential smoothing sono basati sulla descrizione del trend e della seasonality; siamo passati poi ai modelli autoregressivi nei quali il forecasting effettuato in base all'autocorrelazione. I metodi testati sono stati: ARIMA e SARIMAX, abbiamo però escluso subito il primo per via dei risultati, dato che questo modello non è adatto a catturare la seasonality

SARIMAX

L'algoritmo SARIMAX è stato impostato con i parametri order = (2,0,0) e seasonal_order = (2,1,0,24). I valori 2 e 0 sono stati scelti per via dei plot dell'ACF e del PACF, mentre 24 è stato scelto per descrivere un periodo di seasonality pari a 24 record ossia un giorno.

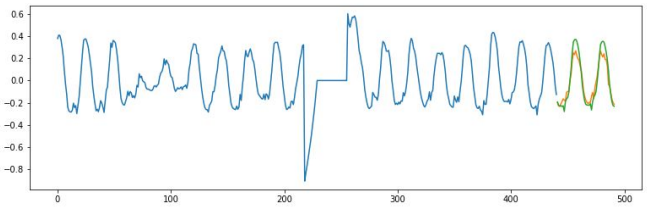


Figura 31: SARIMAX

I risultati ottenuti sono:

MAE	0.031
RMSE	0.038
MAD	0.024
R2	0.951
MAPE	0.349
MAXAPE	2.837
TAPE	17.815

AIC	-936.591
BIC	-916.146

Tabella 18: prestazioni sarimax method

Siamo riusciti ad ottenere in entrambi i casi buoni risultati ma con l'exponential smoothing otteniamo risultati leggermente migliori, una possibile causa può essere una mancata ottimizzazione da parte nostra dei parametri nel modello SARIMAX. In entrambi i casi è stata usato un train pari a 9/10 della ts per un totale di 453 record su 504.

Time series classification

Per il proposito di questa task abbiamo pensato di assegnare una nuova label target al dataset. In particolar modo è stata assegnata la label "Mesi caldi" (0) per i record appartenenti a mesi primaverili ed estivi e "Mesi freddi" (1) per i record appartenenti a mesi autunnali ed invernali.
La classification è stata condotta su time series settimanali, ricavate dall'attributo C6H6(GT). Sono state condotte diverse prove, in particolare trainando gli algoritmi sui valori normali e sui valori normalizzati (MinMaxNormalization). Sono risultati generalmente migliori, seppur non di molto, i classificatori applicati sui valori standard delle time series, motivo per cui si discutono quelli.
E' stato innanzitutto testato lo shapelet based classifier, per poi passare all'applicazione di KNN e Decision Tree (in quanto i più prestanti nel dataset normale) per gli shapelet distance based (SDBC) classifiers e per i Feature based classifier (FBC).
Riportiamo in formato tabellare i risultati dei vari algoritmi, riportandone le metriche quali accuracy, precision, recall, average F1 e la confusion matrix.

Algoritmo	Accuracy	Precision	Recall	Average F1	Confusion
shapelet based	0.411	0.4	0.5	0.409	[3, 6], [4, 4]
SDBC Decision Tree	0.705	0.615	1	0.688	[4, 5], [0, 8]
SDBC KNN	0.529	0.5	0.5	0.527	[5, 4], [4, 4]
FBC Decision Tree	0.588	0.555	0.625	0.588	[5, 4], [3, 5]
FBC KNN	0.588	0.555	0.625	0.588	[5, 4], [3, 5]

Tabella 19: prestazioni shaped distance based e feature based time series classifiers

come si può notare dalla precedente tabella le prestazioni degli algoritmi non sono troppo soddisfacenti. Gli algoritmi tendono ad effettuare errori nella classificazione con molta facilità,

probabilmente a causa di una non troppo evidente differenza tra le time series dei mesi caldi e dei mesi freddi.
Sono stati poi testati i soliti classificatori (Decision Tree e KNN, utilizzando la distanza sakoechiba) sui training composti dai valori delle time series.
Ne risulta quanto segue:

Algoritmo	Accuracy	Precision	Recall	F1 score	Confusion
Knn	0.88	1	0.75	0.878	[9, 0], [2, 6]
Decison Tree	0.705	0.66	0.75	0.705	[6, 3] [[2, 6]

Tabella 20: prestazioni KNN (sakoechiba) vs DT

I classificatori lavorano molto meglio rispetto al caso precedente, e il knn, utilizzando come metrica di distanza “dtw_sakoechiba”, fornisce una classificazione perfetta per i mesi estivi.

Convolutional neural network

Sono stati effettuati diversi tentativi di modellazione del CNN, tuttavia riportiamo il modello che ha ottenuto le prestazioni migliori.
L’architettura adottata è costituita da 4 convolutional layers di tipo 1D, il primo avente 16 filtri, come input il train set, e come activation function ‘relu’. L’input dei successivi layers sono il risultato dell’applicazione del primo layer, cambiando solo il numero dei filtri: il secondo ha 32 filtri, e activation function “relu”, il terzo ha 64 filtri, e la stessa activation function mentre il quarto ha 128 filtri, e la stessa activation function. La Convolutional Neural Network non è fully connected. Dopodichè è stato aggiunto un layer di output avente due noti e come activation function “sigmoid”. Riportiamo le metriche ottenute e la confusion matrix.

Algoritmo	Accuracy	Precision	Recall	F1 score	Confusion
best CNN	1	1	1	1	[9,0], [0,8]

Tabella 21: prestazioni convolutional neural network

Con tale architettura il convolutional neural network classifica alla perfezione i due tipi di time series.

Recurrent neural network

Per i recurrent neural network sono state testate diverse architetture. Discuteremo in questa sede quella che ha ottenuto le prestazioni migliori.

L’architettura è la seguente: come input layer c’è un LSTM layer a 2 nodi. Dopo il nodo input il primo hidden layer è costituito da un layer di tipo LSTM avente 16 nodi. A seguire un altro LSTM layer avente 64 nodi. Dopo i layer di tipo LSTM sono presenti altri 3 hidden layer normali (di tipo Dense), aventi rispettivamente 128, 64 e 32 nodi. Infine come nodo output vi è un Dense layer a 2 nodi.
Tutti i layer prima dell’output layer hanno come activation function ‘relu’, mentre il layer output ha come activation function “sigmoid”. in tutti i layer il kernel initializer è di tipo “truncated Normal. L’architettura non è fully connected, in quanto è previsto un dropout per ogni hidden layer. Per ogni hidden layer è prevista anche la BatchNormalization. Il modello è stato compilato seguendo come loss function la “sparse_categorical_crossentropy”, come ottimizzatore “adam” e come metrica di riferimento l’accuracy. Il modello è stato trainato su 50 epoche.
Con un’architettura di questo tipo sono stati ottenuti i seguenti score:

Algoritmo	Accuracy	Precision	Recall	F1 score	Confusion
RNN	0.647	1	0.25	0.575	[9, 0], [6, 2]

Tabella 21: prestazioni recurrent neural network

Come si può notare tale struttura presenta performance molto meno valide rispetto ad altri tipi di architetture.
Per formulare qualche considerazione finale circa questa task di time series classification, si può notare come l’architettura migliore in grado di catturare in maniera perfetta le caratteristiche di una time series al fine di classificare è il convolutional neural network, probabilmente perchè lo strumento più adatto nella classificazione delle immagini. La time series infatti rimane visivamente molto intuitiva a livello grafico, e probabilmente proprio sulla base di questo questo tipo di rete neurale è particolarmente efficiente.

Multivariate time series classification

Per il proposito di questa task abbiamo provato principalmente ad applicare Recurrent Neural Networks e Convolutional Neural Network su diverse combinazioni di attributi. Discutiamo in questa sede i risultati ottenuti dai due modelli più prestanti.
Il primo modello è composto dagli attributi C6H6(GT), CO(GT), NOx(GT) in quanto più significativi per la PCA. La variabile target è sempre stata assegnata con “Stagioni calde” e
“Stagioni fredde” come avvenuto nelle classificazioni precedenti.
Con questo set di attributi il modello migliore è stato ottenuto con una Convolutional Neural Network composta di 5 convolutional layers composti di rispettivamente 256, 128, 64, 32, 16 filtri, aventi come activation function “relu”, seguiti da un Dense layer di due nodi con activation function “sigmoid”. Come optimizer è stato mantenuto “adam”, come metrica di riferimento “accuracy” e come loss “sparse categorical crossentropy”. La rete neurale non è completamente connessa.
Il modello è stato allenato su 150 epoche, ottenendo le seguenti prestazioni:

Algoritmo	Accuracy	Precision	Recall	F1 score	C. Matrix
CNN	1	1	1	1	[9, 0], [0, 8]

Tabella 22: prestazioni multivariate convolutional neural network

Come si può notare, le prestazioni ottenute sono a dir poco soddisfacenti, risultando una classificazione perfetta.

Multivariate Recurrent Neural Network

Il miglior modello ottenuto dal recurrent neural network è stato ottenuto con gli attributi C6H6(GT), CO/GT), NOx(GT), PT08.S3(NOx).
Il modello è composto di 3 layers LSTM, con rispettivamente 256, 128, 64 nodi, e activation function "relu". A questo seguono 3 hidden layers di tipo Dense con 256, 64, 32 nodi e come activation function sempre la stessa.Segue un layer Dense in output con due nodi e come activation function "sigmoid". La rete neurale non è completamente connessa. L'ottimizzatore è settato su "adam", come metrica di riferimento si ha "accuracy" e come loss function "sparse categorical crossentropy. il modello è stato allenato su 50 epoche, ottenendo i seguenti risultati:

Algoritmo	Accuracy	Precision	Recall	F1 score	C. Matrix
RNN	0.822	1	0.75	0.878	[9, 0], [2, 6]

Tabella 23: prestazioni multivariate recurrent neural network

Come si può notare le prestazioni di questo algoritmo sono abbastanza buone. Rimangono degli errori di classificazione per quanto riguarda la classe 1.
Per concludere possiamo notare come il convolutional neural network sia decisamente efficace per risolvere la time series classification task.

TASK 4 - Sequential Pattern Mining

Durante lo svolgimento di questa task abbiamo utilizzato la time series dell'attributo temperature dal 2004-06-21 al 2004-09-20, osserviamo quindi l'andamento della temperatura durante il corso del periodo estivo. I valori selezionati per i parametri *n_segments* e *alphabet_size_avg* della sax transformation sono stati scelti in modo tale da ottenere una trasformazione il più fedele possibile alla time series originale con il minor numero di segmenti possibili. Dopo diversi tentativi nostra time series a seguito della trasformazione è composta da 180 segmenti e vengono usati 11 simboli.

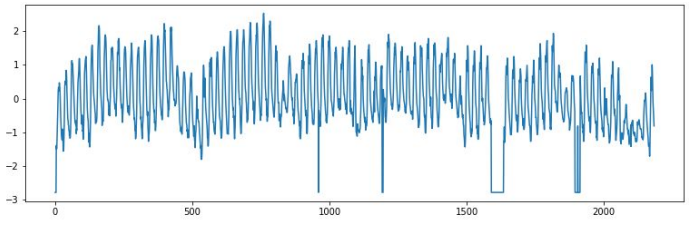


Figura 32: Time series prima della segmentazione

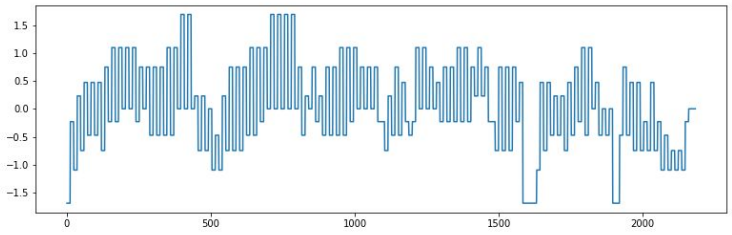


Figura 33: dopo la segmentazione

Successivamente abbiamo suddiviso la time series in settimane per analizzare i pattern più frequenti considerando: una transazione pari ad una settimana. Per ottenere questa impostazione creiamo 15 time series di lunghezza 12. Il numero 15 è stato scelto effettuando una proporzione in modo da ridurre il ciclo di una settimana al numero di segmenti della time series ottenuti dopo la sax transformation. Otteniamo i seguenti pattern frequenti:

Support	n_pattern
2	436
3	243
4	119
5	57
6	29
7	15
8	10
9	5
10	3

Tabella 24: support e numero pattern

Numero pattern presenti per lunghezza e frequenza:

	Lunghezza pattern							
	3	4	5	6	7	8	9	10
support								
2	93	106	73	48	31	15	10	3
3	81	72	28	7				
4	48	20	2					

5	19	1						
6	3							

Tabella 25: support e lunghezza pattern

Riportiamo graficamente i pattern che riteniamo più interessanti. I tre pattern che hanno ottenuto il support maggiore (6):

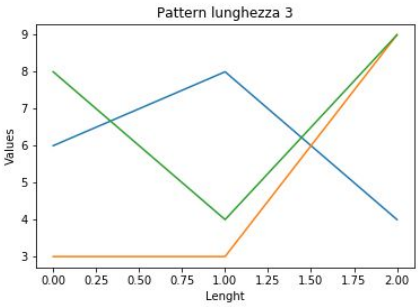
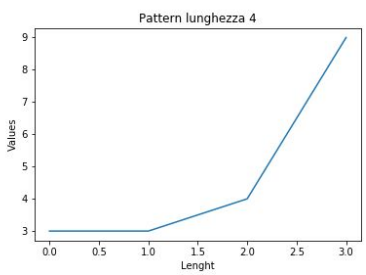


Figura 31: pattern di lunghezza 3

I pattern di lunghezza 4 e 5 con supporto rispettivamente pari a 5 e 4:



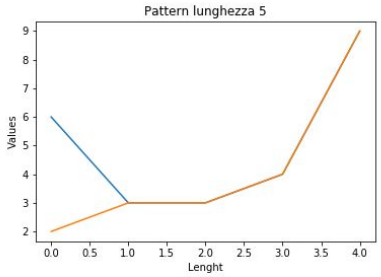


Figura 35: pattern di lunghezza 4 e 5

Possiamo osservare che il pattern di lunghezza 4 è presente anche nel pattern frequente di lunghezza 5.

Task 5 - Outliers detection

Prima di analizzare il dataset in cerca degli outliers maggiori ho ridotto il numero di attributi secondo le valutazioni già effettuate nella prima task e ho eliminato i NaN e gli outliers dovuti ad errori di battitura (i valori pari a -200 presenti in tutte le distribuzioni)

Per la classificazione è stata presa come variabile target ‘weekend’ e isolato dal dataset iniziale dei record per il train e test.

Per individuare gli outliers di ogni distribuzione ho visualizzato i boxplot e creato un dataset di soli outliers da analizzare con gli algoritmi di outliers detection.

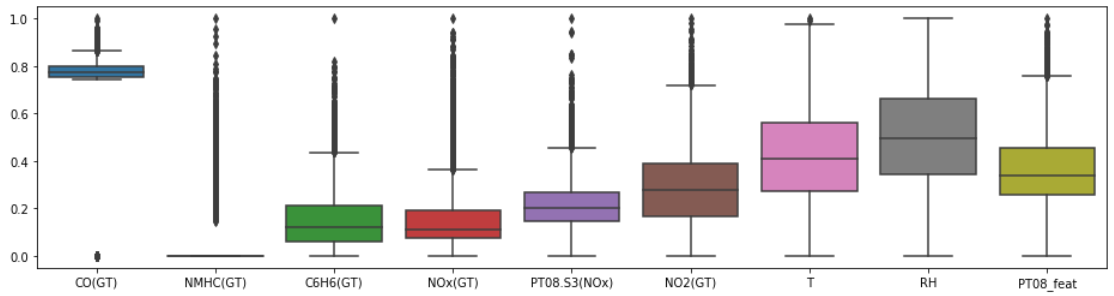


Figura 36: boxplot attributi

DBSCAN

Il primo approccio utilizzato per ricercare gli outliers che più si discostano visivamente da tutti gli altri record è il DBSCAN, per ricercare i parametri ottimali ho utilizzato il knee method che ha evidenziato un epsilon ottimale di 0.18 e il valore di min.pts è il risultato del logaritmo in base n del numero dei record, uguale a 9.
Con i risultati ottenuti da questo metodo ha evidenziato come outliers tutti i record classificati con label pari a -1, in totale 240.

Local Outlier Factor

Con il metodo LOF un valore outlier si discosta dagli altri per la densità di altri punti intorno ad esso, minore rispetto alla densità di valori nei dintorni di un valore normale. L'Outlierness Score in questo caso è il 'negative outlier factor', quindi più basso è questo Score e maggiore è l'Outlierness del punto.
Con SKLearn è stato settato l'Iper-Parametro 'n neighbors' pari a 100, in modo da evidenziare solamente i top outliers all'interno del dataset e anche per il minor numero di prediction errors riscontrati (51) . Il metodo restituisce 168 Outliers, identificati come i valori che si discostano maggiormente dallo score di -1 attribuito ai valori normali. In questo caso la soglia per cui un valore è considerato outlier dal modello è di -1.5 circa, con il valore che più si scosta dalla distribuzione pari a -3.40.

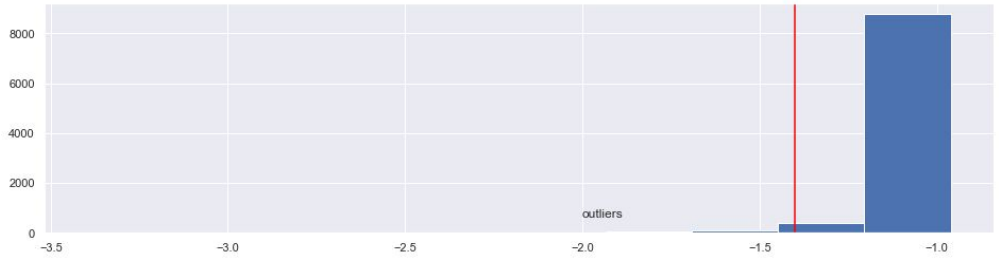


Figura 37: lof treshold

Il secondo grafico mostra lo scatter plot degli attributi "C6H6" e "PT08_feat" accerchiati da una circonferenza con raggio proporzionale al LOF Score assegnato dal modello.

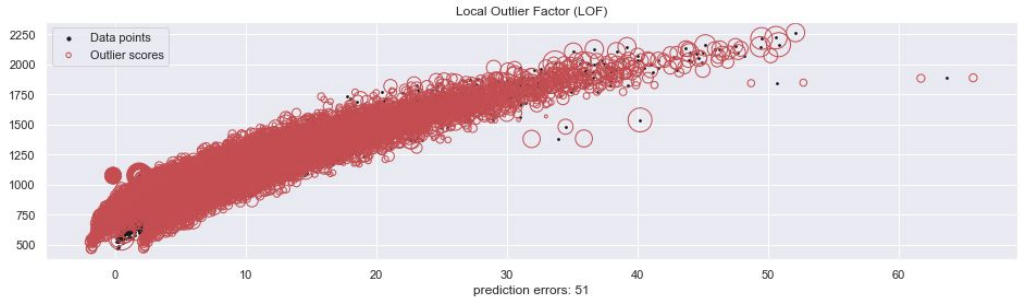


Figura 38: scatter plot con lof score

ABOD

E' stato scelto 50 come valore dell'Iper-Parametro 'neighbor' in quanto per valori più bassi il modello non riusciva ad identificare alcun outlier. L'algoritmo così impostato ha identificato 8430 Inlier e 927 Outlier, assegnando un punteggio di Outlierness ad ogni punto (oggetti con punteggi alti sono considerati Outliers). Il punteggio assegnato agli oggetti del dataset varia nell'intervallo fra -14.86 e -1.058 ed il punteggio sopra cui i valori sono considerati Outliers è -3.743364875.

KNN

Il K-Nearest Neighbors individua gli outliers calcolando le distanze di ogni record dal suo 50-NN, in modo da ottenere nel nostro caso sono stati identificati 757 outliers sulla base di una threshold pari a 65.63.

Come mostrato dal grafico la maggior parte dei record ha uno score compreso tra 20 e 50, lo score medio degli outliers è di 65.65 con un valore massimo di 481 .

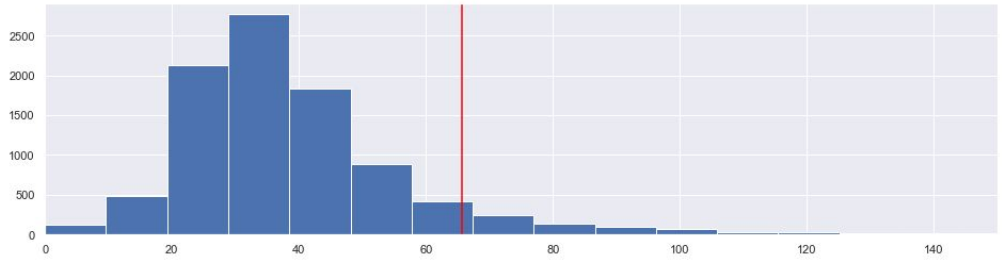


Figura 39: knn treshold

AutoEncoder

Il modello è stato costruito con numero di nodi [25, 2, 2, 25] ed istruito con 50 epoche. Sono stati identificati 936 outliers. Nel primo grafico sono state rappresentate le distribuzioni di frequenza della probabilità che ciascun punto sia un Outlier (istogramma blu) e la probabilità inversa, cioè quella che un certo oggetto sia un Inlier (istogramma arancione).

Il secondo grafico è la distribuzione di probabilità del punteggio di Outlierness assegnato dal modello.

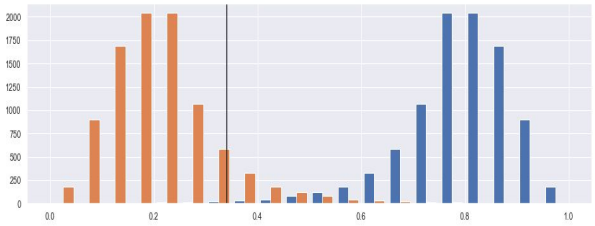


Figura 40: distribuzione di probabilità classificazione

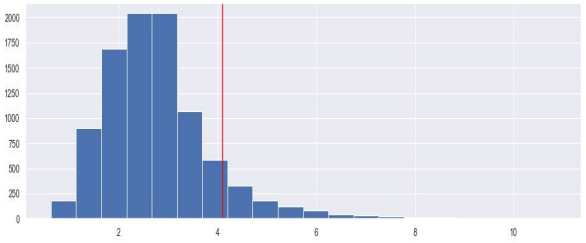


Figura 41: autoencoder treshold

Scatter plots

In conclusione all'Outlier Detection viene mostrato lo scatter plot derivato dalla riduzione dei record tramite PCA al fine di rendere più comprensibile l'immagine, con evidenziate gli outlier rilevati da ogni modello (colore verde)

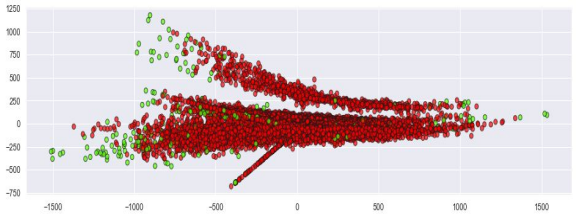


Figura 42: scatter plot dbscan

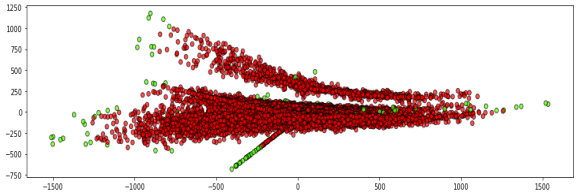


Figura 43: scatter plot lof

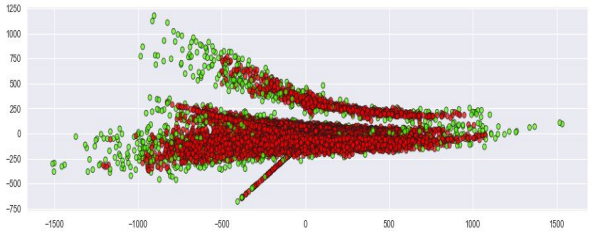


Figura 44: scatter plot knn

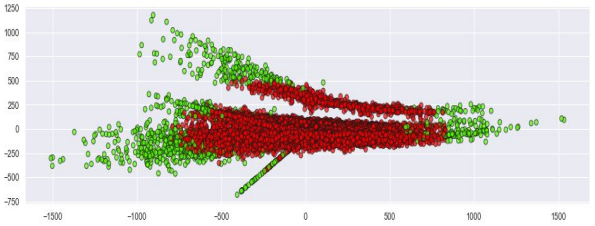
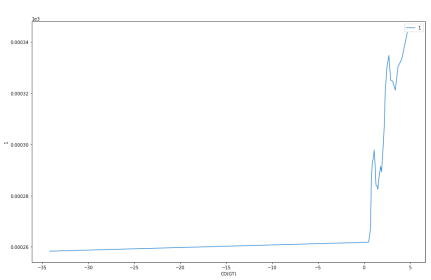
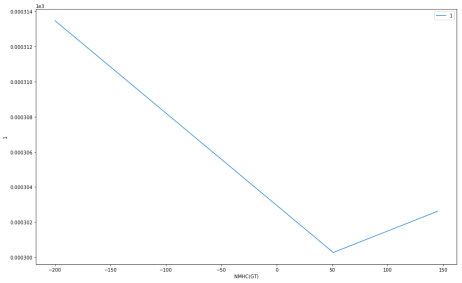


Figura 45: scatterplot autoencoder

Explainability

Per rendere più chiare le classificazioni nelle task precedenti vengono qui riportati dei grafici di dipendenza parziale della classificazione delle singole feature più importanti. Come classificatore è stato utilizzato il Random Forest perchè tra i più performanti.





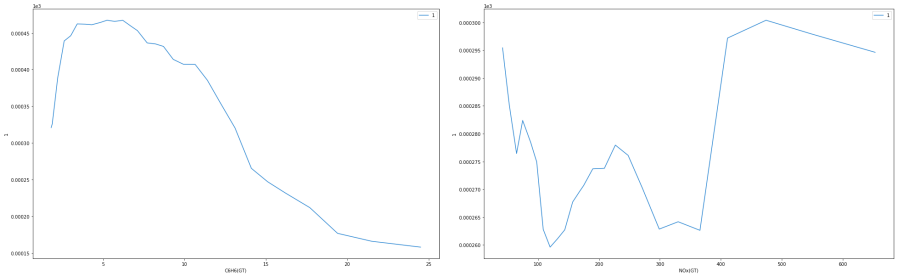
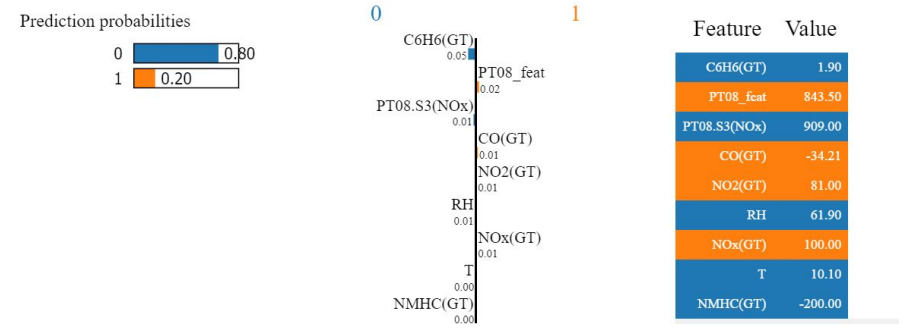


Figura 46: grafici dipendenza parziale

E’stato utilizzato il metodo *LIME* per spiegare la classificazione effettuata su un record estratto casualmente dal dataset, in questo caso il numero 550.



In figura possiamo osservare i risultati del metodo. A destra abbiamo le probabilità che l’oggetto sia di classe 0 (80%) o di classe sia 1 (20%). Al centro abbiamo la divisione degli attributi tra quelli che prevedono una classe 1 per l’oggetto (*PT08_feat, CO(GT), NO2(GT), NOx(GT)*) e gli altri che prevedono una classe 0. Possiamo notare quanto sia marginale l’importanza di tutti gli attributi, anche i primi due per feature importance. Quì ad ogni attributo è anche associato il valore della sua Feature Importance nella classificazione. In fine, l’ultima figura rappresenta i valori che assume l’istanza in questione per ogni attributo.