Lez2_

October 27, 2023

1 Lezione 2 (Russo Russo)

1.1 Parte codice end2end.ipynb

1.1.1 Cosa fare in caso di valori mancanti?

Valori mancanti: 1. elimino riga o colonna 2. stimo valore mancante, ad esempio con una media

Alcune osservazioni su snippet di codice

[3]:
```
# Scikit-Learn  0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# to make this notebook's output identical at every run
np.random.seed(42)

DATASET_FILENAME="housing.csv"
DATASET_DIR="./data/"
DATASET_URL = "https://raw.githubusercontent.com/ageron/handson-ml2/master/
 ↪datasets/housing/housing.tgz"

def fetch_housing_data(housing_path=".", housing_url=DATASET_URL):
    import tarfile
    import urllib.request
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
```

```
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()

if not os.path.exists(os.path.join(DATASET_DIR, DATASET_FILENAME)):
    fetch_housing_data(housing_path=DATASET_DIR)
import pandas as pd

housing = pd.read_csv(os.path.join(DATASET_DIR, DATASET_FILENAME))
housing.head()
```

[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	

	population	households	median_income	median_house_value	ocean_proximity
0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	496.0	177.0	7.2574	352100.0	NEAR BAY
3	558.0	219.0	5.6431	341300.0	NEAR BAY
4	565.0	259.0	3.8462	342200.0	NEAR BAY

1.1.2 Osservazioni

`'Colonna ocean_proximity': valori categorici.`

Algoritmi di ML non lavorano su questi valori, ma su valori numerici. Possibili soluzioni: - Associare un numero ad ogni categoria (OrdinalEncoder). - OneHotEncoder, classifica categorie con valori binari. Creo un attributo per ogni categoria, l'attributo vale sempre 0, tranne nella categoria corrispondente. Ad esempio bianco, rosso e verde, se macchina è rossa ho 1 solo in corrispondenza dell'attributo rosso.

sklearn consente di mettere insieme tutti gli step che abbiamo visto in pipeline, così da automatizzare tutti questi step. Metodo fit_transform per appricare la pipeline.

[9]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

#num_pipeline = Pipeline([
 #       ('imputer', SimpleImputer(strategy="median")),
  #      ('attribs_adder', CombinedAttributesAdder()),
   #     ('std_scaler', StandardScaler()),  # Feature scaling
    # ])
#housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Scaling: ogni attributo ha valori su un certo range, avere features su scale di valori diversi crea problemi, quindi buon approccio è normalizzare tutte le featrues tra 0 e 1.

Una volta che il dataset è pronto (*preprocessing terminato*) devo andare ad addestrare un modello di ML. Devo scegliere modello, per farlo devo usare la conoscenza che ho dei modelli per scegliere quale provare.

Esempio con **linearRegression**: instanzio la classe e poi uso fit.

Tutto i modelli hanno due metodi: fit e predict: - fit per addestrare, a cui passo il dataset preparato con preprocessing e le etichette. - predict per predire, quindi ottengo le predizioni.

Dato che ho definito pipeline di preprocessing sul training set, devo eseguirla anche sui dati che voglio predire. Poi uso predict e mi dà predizioni. Per valutarli stampo anche i target e vedo quanto il modello ha sbagliato. Non si può valutare errore a occhio, ma devo valutare delle metriche: **errore quadratico medio**: gli passo predizioni ed etichette calcolate su tutto il dataset.

[ ]:
```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
```

A questo punto posso provare a cambiare features oppure modello. Per cambiare modello devo solo importare un'altra classe e la devo addestrare come fatto per la regressione lineare.

1.1.3 Valutazione del modello

Proviamo ad usare un **decision tree**, errore viene 0.0. Questo ci fa "insospettire", probabilmente c'è *overfitting*. (ovvero modello si è adattato troppo al training set, e quindi sul testing set farà male)

Come faccio ad avere una stima migliore del comportamento del modello? Devo usare *validation set*, in particolare si può usare *cross-validation*

Perchè si usa il validation set invece del test set?

- Non è detto che l'ingegnere che deve trovare soluzione del problema di ML abbia un test set.
- Valutare l'errore su quanto si comporta bene sul test set è sbagliato, perchè potrei avere modello che casualmente si comporta meglio sul test set, ma che in generale non si comporta così bene. Le scelte devono basarsi su ciò che vedo prima del testing set. #### Cross validation Idea: Invece di mettere da parte dei dati del training set per usarli nel validation set una sola volta, lo faccio più volte. La funzione già esiste:

`cross_val_score(modello, dati, target, metrica da valutare, quante volte ripetere processo di tagliare il Training set).`

[ ]:
```
from sklearn.model_selection import cross_val_score
```

```
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

Da questo emerge che il decision tree effettivamente aveva errore (simile a quello della regressione lineare), quindi avevamo un caso di overfitting. Emerge che linear regression si comporta meglio facendo cross validation anche con questo modello. Faccio lo stesso anche con **random forest** (forest_reg è la classe da instanziare). Ci mette più tempo per fare training rispetto agli altri due modelli visti. L'errore medio è sceso rispetto agli altri due modelli. A questo punto potrei scegliere random forest e andarlo ad usare sul test set.

Iperparametri Ogni modello ha degli iperparametri, quindi invece di cambiare direttamente modello posso modificare questi iperparametri, devo fare *tuning* degli iperparametri per vedere se sto usando gli iperparametri giusti.

Esempio Random forest visto prima con 100 alberi (100 estimatori), nessuno mi dà garanzie che sia il migliore, quindi si fa a tentativi. Ci sono dei modi per esplorare le varie configurazioni e trovare la migliore. C'è anche **grid search**.

[ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

In random forest abbiamo numero di estimatori e numero di features in base a cui fare lo splitting, con questi due attributi costruisce una griglia con tutte le possibili combinazione, addestra il modello con tutte le configurazioni e vedo i diversi errori, poi scelgo la migliore. Devo scegliere io quali iperparametri usare nella griglia, quindi avrò tutte le possibili combinazioni relativamente a features ed estimatori che ho deciso di mettere nella griglia.

Con `GridSearchCV`→ classe che fa grid search con cross validation. Anche un modello semplice usato con grid search, tanti dati e cross validation diventa impegnativo. Per ridurre complessità invece di fare tutto ciò posso usare: `RandomizedSearchCV`. Può dare risultato peggiore, ma è più veloce. Ci sono anche altri modi, invece di pescare gradualmente se cella della griglia è andata bene devo esplorare i dintorni di quella configurazione. Alla fine uso il metodo `best_estimator`, dopo aver fatto grid search, per vedere qual è la migliore configurazione degli iperparametri.

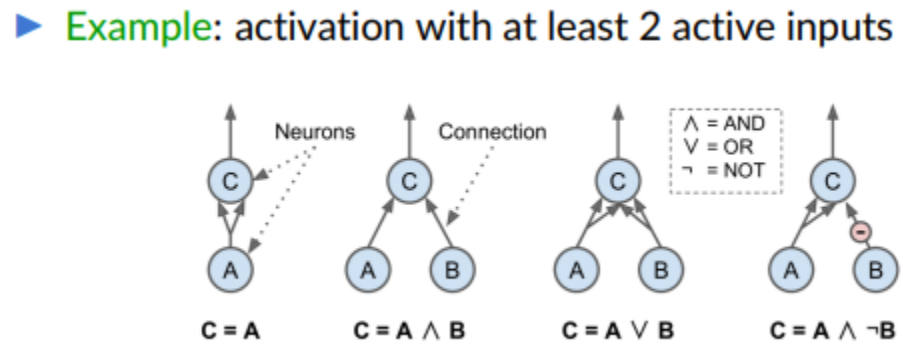
1.2 Parte di teoria

1.2.1 Reti neurali e deep learning:

Reti neurali cercano di simulare funzionamento dei neuroni. Questo parallelismo è stato origine ispirazione, però di fatto la direzione presa non ha nulla a che vedere con il cervello umano.

1.2.2 Prima rete neurale artificiale 1943: McMulloch e Pitts.

Input neuroni binario = avere o non avere impulso elettrico Output attivato quando almeno X input sono attivi. Nell'esempio su slides X=2.



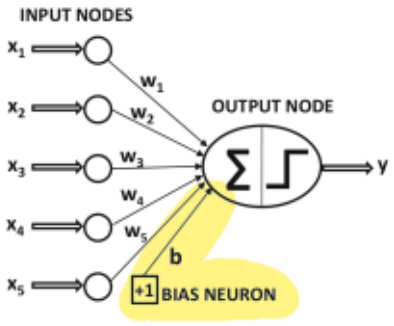
1.2.3 Perceptron

Modello del 1957. No input/output binari, ma numeri reali. Input collegati al perceptron tramite dei pesi, che sono numeri reali, quindi ogni input è moltiplicato per il suo peso. Cosa fa il perceptron con questi input e pesi? *somma tutti gli input pesati*; questo risultato viene passato attraverso la funzione di attivazione, che decide qual è l'output del perceptron. Questa funzione di attivazione è la funzione segno nel modello originario. (risultato positivo uscita 1, risultato negativo -1).

1.2.4 Perceptron with bias

Aggiungo termine di bias, a cosa serve? Supponiamo di voler costruire modello che vuole fare predizioni, ad esempio qualsiasi siano gli input devo predire sempre il valore 5. Se tutti gli input sono 0 come faccio a predire 5? Impossibile, quindi deve esserci una componente dell'uscita che non deve dipendere dall'input, questo è il termine di bias. E' come se avessi input aggiuntivo pari a 1 sempre, mentre il peso è b.

- There is often an invariant part of the prediction, called bias
  - e.g., you need to predict a positive value when $x_j = 0 \; \forall j$
- We consider an additional input node that always transmits a constant value 1 with connection weight b (bias variable)



$$y = \phi(\sum_{j=1}^{d} w_j x_j + b) = \phi(\mathbf{w}^T \mathbf{x} + b) \qquad (2)$$

Il perceptron così definito è un *classificatore binario*. E' come se avessi retta che divide in due parti il piano, risultato cambia se sta sopra o sotto la retta, la posizione e la pendenza della retta dipendono da input e pesi.

1.2.5 Come si addestra questo modello?

1. Inizializzare i pesi a 0
2. Fornisco istanza in input e vedo che y tira fuori il perceptron.
3. Vedo rispetto a valore target se ha sbagliato.
4. Se ho errore rinforzo le connessioni che possono aiutare a non sbagliare, cambio il peso (e allo stesso modo del bias, che è peso relativo a input sempre pari a 1). Nota: Il perceptron da come predizione "-1" o "+1", che devo confrontare con la soluzione "t". Se toppo, vado a modificare il peso di quella istanza. Se la predizione è corretta, t(i) e y(i) sono uguali, e quindi non cambio il peso. Se predizione è "+1", e sbaglio "-1", aumento peso (1-(-1)) = 2. Nel caso contrario diminuisco il peso.
5. Itero per un certo numero di volte (o finchè non raggiungo qualche criterio)

Esempio con dataset iris (fiori) Voglio solo sapere se iris appartiene a setosa o no, quindi cambio il target, mi dice solo se appartiene alla prima specie o no. Faccio questo perchè il perceptron è un classificatore binario. Istanze della prima sono molto facili da distinguere, si vede dal grafico che stanno tutte raggruppate sotto al valore della $y = 1$; il problema è molto semplice, si risolve a occhio. Basterebbe un if in python a risolverlo. Stampa pesi, bias e accuratezza su training set e testing set (pari a 100%). Posso definire la *decision boundary*, ovvero il confine di decisione di questo perceptron, ovvero trovo la retta, per farlo devo risolvere un'equazione: $x1 \cdot w1 + x2 \cdot w2 + b = 0$ con $x1 = x$ e $x2 = y$. Scrivo tutto in funzione di y e trovo equazione della retta. Trova una retta di *demarcazione*, non è l'unica chiaramente.

1.2.6 Perceptron and SGD

E' stato fatto reverse engineering del perceptron per vederlo in termini di **stochastic gradient descent SGD**. Loss function che vale 1 quando sbaglio predizione, 0 quando è giusta.

- Consider a 0-1 loss function for the instance $(x^{(i)}, t^{(i)})$

$$\mathcal{L}_{0/1}^{(i)} = \frac{1}{2}(t^{(i)} - sgn\{w^T x^{(i)}\})^2 = (1 - t^{(i)} sgn\{w^T x^{(i)}\}) \quad (5)$$

0 ok, 1 se sbaglio. Lavoro svolto dal Perceptron mediante vettori "x" e "w trasposto"

La derivata della funzione segno ha un problema, in tutti i punti è 0, problema per applicare metodo del gradiente; quindi viene sostituita con altra funzione di loss più trattabile, ma che minimizzata è stessa cosa.

- We consider a smoothed surrogate loss function:

$$\mathcal{L}^{(i)} = \max\{0, -t^{(i)}(w^T x^{(i)})\} \quad (6)$$

Usiamo una funzione "surrogata", che prende il massimo tra i due valori proposti. L'idea è che se t >0, allora segno è negativo, e vince 0. Se t<0, col segno meno ottengo un valore >0, e quindi assumiamo questo valore come risultato. Graficamente, questa funzione non è troppo diversa dall'originale, infatti ne mantiene lo stesso punto di minimo.

perceptron-sgd.ipynb

12

Ottengo stesso valore per il peso che usando la loss segno. Calcolo derivata della loss rispetto al valore dei pesi. Ottengo formula per aggiornamento di w, se metto il parametro eta pari a 1 riottengo la formula del perceptron.

$$\mathcal{L} = \sum_{i=1}^{N} \max\{0, -t^{(i)}(w^T x^{(i)})\} \quad (7)$$

- We can compute the gradient and the SGD update of the smoothed surrogate loss function:

$$\nabla_w \mathcal{L}^{(i)} = (y^{(i)} - t^{(i)}) x^{(i)} \quad (8)$$

$$w \leftarrow w - \eta \nabla_w \mathcal{L}^{(i)} = w + \eta(t^{(i)} - y^{(i)}) x^{(i)} \quad (9)$$

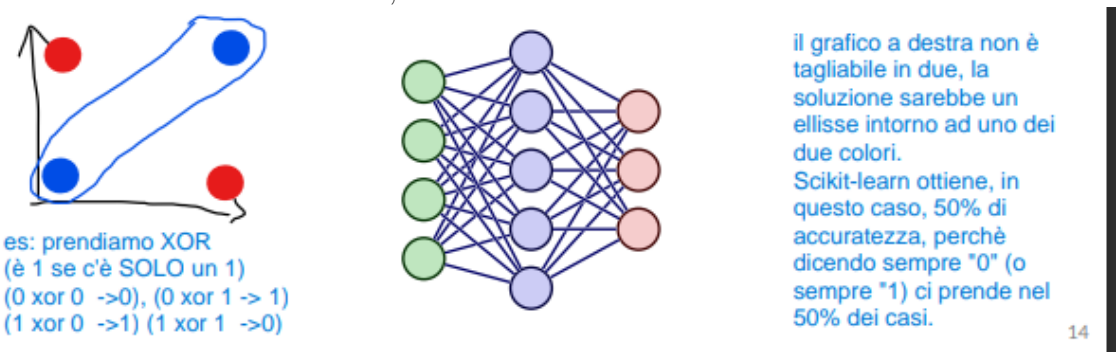
With $\eta = 1$, we get back the Perceptron update.

Questa variabile (eta) rappresenta il "learning rate". Se eta = 1, torniamo al caso originale del Perceptron.

1.2.7 Beyond the Perceptron

Il perceptron è una rete neurale artificiale con un solo livello computazionale, quindi rete molto semplice. Per come è fatto questo modello può solo apprendere delle decision boundaries lineari.

Esempio perceptron per xor Non esiste retta per dividere spazio tra le due categorie. Questa è stata delusione di chi lavorava con questi modelli, che possono apprendere solo funzioni lineari. Trova pesi 0 e bias 0, mentre l'accuratezza è 50%, dicendo sempre 0 o sempre 1, comunque al 50% ci prendo. Quindi cosa si fa? L'idea interna del perceptron usata nelle reti neurali artificiali, inevce di un solo neurone si crea rete di neuroni interconnessi.



[ ]: