SAPIENZA
UNIVERSITÀ DI ROMA

Student face detection, flow prediction and classroom optimal allocation

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Informatica

Candidato

Francesco Zappia
Matricola 1757366

Relatori

Roberto Capobianco
Lavinia Amorosi

Anno Accademico 2018-19

Student face detection, flow prediction and classroom optimal allocation
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: zffromGerace@gmail.com

Sommario

Questa tesi è incentrata sullo sviluppo di un'applicazione da utilizzare in ambito universitario atta ad automatizzare il processo di raccolta dati riguardanti il flusso di studenti in particolari circostanze (temporali ed accademiche) utilizzati a loro volta per stimare il numero di allievi presenti in nuove contesti; i risultati vengono in seguito impiegati per assegnare le diverse classi a set di aule rispettivamente con capienze adatte a soddisfare la loro richiesta ottimizzando la loro allocazione in base a specifiche ed ulteriori richieste.

Contents

1 Introduzione **1**
1.1 Il problema . . . 1
1.2 Struttura del documento . . . 1
1.3 Codice sorgente . . . 3

2 State of the art **5**
2.1 Face Detection . . . 5
2.1.1 Problematiche comuni . . . 5
2.1.2 Approcci feature-based . . . 5
2.1.3 Algoritmo Viola-Jones per la Face Detection . . . 7
2.2 Regressioni e Machine Learning . . . 9
2.2.1 Reti neurali . . . 9
2.3 Programmazione lineare . . . 12
2.3.1 Programmazione lineare multiobiettivo . . . 12
2.3.2 Programmazione lineare intera multiobiettivo . . . 13
2.4 Lavori precedenti . . . 15

3 Metodi **17**
3.1 Individuazione degli studenti . . . 17
3.2 Il modello di regressione . . . 19
3.2.1 Struttura della rete . . . 19
3.2.2 Training . . . 20

Chapter 1

Introduzione

1.1 Il problema

In ambito universitario è problematica molto comune quella di cercare di assegnare aule a gruppi di studenti in modo tale da rispettare le richieste in numero di posti di ciascuno di essi senza nemmeno concretamente conoscere il numero di allievi che frequenteranno realmente il corso in questione, dovendo l'istituto assegnarle in anticipo rispetto all'inizio delle lezioni.

Si pone quindi l'ulteriore problema di facilitare la raccolta di dati che permetta di effettuare delle stime più accurate, quindi di registrare in modo automatico la partecipazione degli universitari a ciascuna lezione e quindi inferire attraverso di essi queste stime.

Quest'applicazione si pone l'obiettivo di risolvere esattamente questa problematica (il funzionamento è illustrato in 1.1): a partire da foto (che si intende essere di aule) riconosce il numero di volti e quindi di studenti in essa presenti. Questa informazione, associata ad informazioni riguardanti la lezione in questione, viene fornita insieme a quest'ultime ad uno stimatore che, in base ai dati ricevuti cerca di costruire un modello matematico capace di stimare il flusso di allievi in nuove e richieste situazioni: delle stime che verranno impiegate per distribuire ciascun gruppo nelle diverse aule con data capacità in modo ottimale, cercando cioè di massimizzare il valore di una certa funzione matematica che rappresenta il problema di allocazione.

1.2 Struttura del documento

Il documento è strutturato in

- **State of the Art**, in cui vengono accennati i concetti teorici utilizzati per la risoluzione del problema; vengono analizzati articoli e fonti che hanno affrontato le stesse problematiche.
- **Metodo**, nel quale sono approfondite le metodologie usate per la realizzazione dell'applicazione e vengono trattate ulteriori questioni sorte nell'implementazione.
- **Risultati**, che illustra gli esiti e l'efficienza della realizzazione scelta.

Numero di studenti

Stimatore

Info foto

Stime

Info aule

Distribuzione classi

Figure 1.1. Il flusso di funzionamento dell'applicazione

- **Conclusioni**, contenente le osservazioni finali dedotte dai risultati ottenuti.

1.3 Codice sorgente

Il codice completo di questa applicazione è accessibile su GitHub al seguente indirizzo

`https://github.com/morpheusthewhite/celephais/`;

analogamente è stato pubblicato il codice LaTeXdi questa tesi al link

`https://github.com/morpheusthewhite/bachelor-thesis`.

Chapter 2

State of the art

In questo capitolo saranno inizialmente esposti i concetti teorici alla base dell'applicazione sviluppata: dagli algoritmi più comuni per la **Face Detection** ad un'introduzione al **Machine Learning** per finire poi con elementi di **Programmazione Lineare Multiobiettivo**.

Verranno quindi analizzati precedenti progetti che hanno affrontato le stesse problematiche.

2.1 Face Detection

La Face Detection è una tecnologia che permette di localizzare ed estrarre da un'immagine la regione contenente un volto [DDB15]; ha oggigiorno diverse applicazioni ed utilizzi, dal video coding al content-based image retrieval.

2.1.1 Problematiche comuni

Le sfide che una tecnica di face detection si trova ad affrontare sono solitamente ([DDB15]):

- **Occlusione**. Spesso parte dei volti sono nascosti da oggetti e/o altri volti
- **Espressioni**. L'aspetto dei volti è fortemente influenzato dall'espressione della persona.
- **Posa**. La posizione da cui è scattata l'immagine influenza la prospettiva del volto (2.2).
- **Luminosità**. Livelli di luminosità eccessivi o troppo bassi impediscono di ricoscere contorni e linee del viso.

2.1.2 Approcci feature-based

In un periodo di ricerca che coinvolge circa gli ultimi trenta anni sono state usati diversi approcci che permettessero di sviluppare tecniche di face detection (2.1), dividendosi principalmente in due categorie: **Feature-based** e **Image-based**.

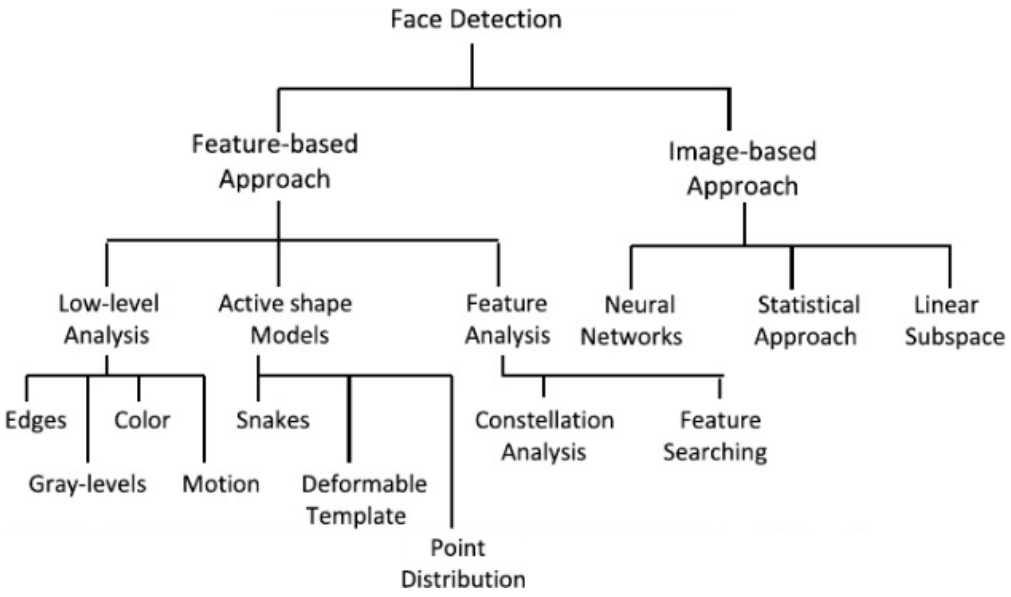

Figure 2.1. Diversi approcci alla face detection

Figure 2.2. Un classico esempio di problema di face detection

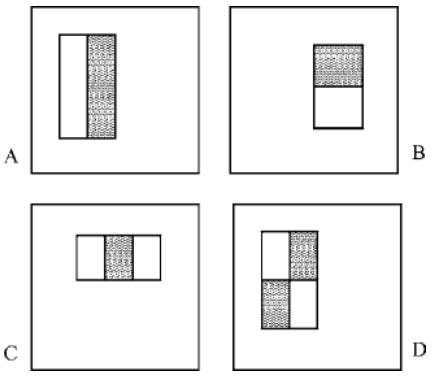

Figure 2.3. Al valore dei pixel nella regione bianca viene sottratto il valore nella regione scura

Nei primi vengono solitamente estratte le feature di un volto da un'immagine (come ad esempio occhi, labbra, sopracciglia) per poi verificare, attraverso la relazione che persiste tra di loro, la presenza di un volto.

All'interno di questa categoria possono essere indivuate ulteriori distinzioni ([DDB15], 2.2):

- **Low Level Analysis**. Feature vengono estratte basandosi sulle proprietà dei pixel come ad esempio colore o valore nella scala di grigi.
- **Feature Analysis**. Vengono sfruttate le proprietà geometrice del volto per cercare di localizzare ed individuare le diverse parti del viso.
- **Active shape models**. Questi modelli, che vanno da *snakes* ai PDM (point-distributed models) sono usati per l'estrazione di feature complesse e per tracciare le iridi degli occhi e le labbra.

2.1.3 Algoritmo Viola-Jones per la Face Detection

Invece tra gli svariati approcci **Image-based** uno che ha avuto particolare successo, soprattutto per la velocità di calcolo che permette il suo utilizzo in applicazioni che utilizzano grandi moli e flussi di immagini, è l'algoritmo Viola-Jones.

Esso fa uso di Haar functions per il l'individuazione delle features, i.e. si basa sul calcolo della somma e differenza di pixel in particolari rettangoli [VJ04], come mostrato in 2.3.

Il calcolo viene enormemente velocizzato attraverso le *integral images*, immagini in cui il valore dei pixel è pari alla somma dei pixel in alto a sinistra, i.e.

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.1}$$

con $ii(x, y)$ il valore del pixel nella *integral image* e $i(x', y')$ il valore nell'immagine originale.

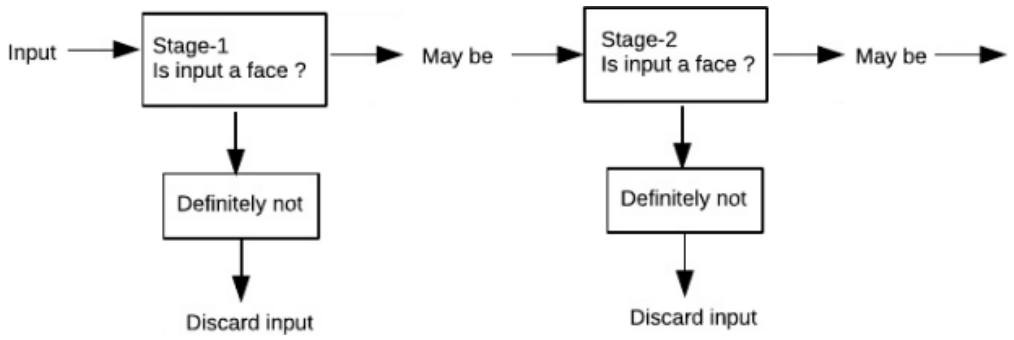

Figure 2.4. Struttura dei classificatori a cascata

Tali *Haar functions* vengono usate quindi da **classificatori**, delle funzioni che mappano un'osservazione (in questo caso un insieme di pixel) su un un set finito di valori, in questo caso [Wan14] $f : \mathbb{R}^d \to \{-1, 1\}$.

Attraverso un Ada Boosting questi singoli ed imprecisi classificatori vengono combinati tra di loro per ottenere un classificatore "forte" che risulta essere molto più accurato [Sch13] dei singoli.

Dato un esempio da classificare x, l'esito della classificazione "forte" è pari a

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{2.2}$$

con $h_t(x)$ classificatore "debole" a cui corrisponde un peso "di voto" pari a α_t: il risultato è quindi ottenuto come la maggioranza pesata delle classificazioni deboli.

Il rendimento può essere ulteriormente migliorato con l'utilizzo di classificatori a cascata (2.4). Un classificatore a cascata è costituito da diversi *stages*, ognuno contenente uno classificatore "forte", ciascuno dei quali determina se la finestra in input non contiente sicuramente un volto oppure potrebbe. Quando viene predetta l'assenza certa di un viso l'immagine viene scartata, altrimenti è passata allo *stage* successivo [DDB15].

2.2 Regressioni e Machine Learning

La regressione è una tecnica statistica utilizzata per determinare la relazione che persiste tra due o più variabili: essa è principalmente adoperata per predizioni e inferenze.

Nella sua forma più semplice (bivariata) la regressione mostra la relazione tra una variabile indipendente X ed una variabile dipendente, Y, attraverso un'equazione del tipo

$$Y = \beta_0 + \beta_1 X + u \tag{2.3}$$

La regressione riesce quindi a quantificare quanto la variazione di una variabile è dipendente dalla variazione di un'altra; quello che invece la non è capace di dimostrare è la casualità, la quale è dimostrata solo analiticamente [CC08].

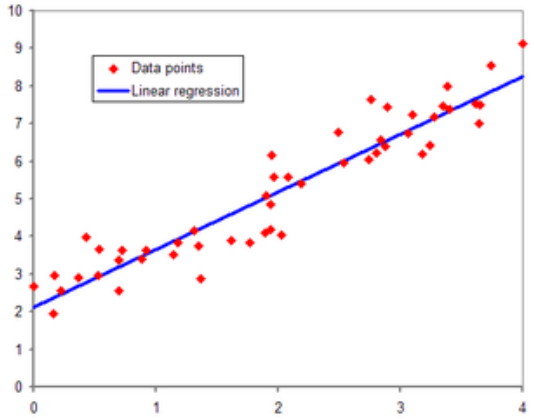

Figure 2.5. Un esempio di utilizzo di regressione lineare: la retta cerca di stimare la relazione tra le due grandezze raffigurate

2.2.1 Reti neurali

Negli ultimi decenni il Machine Learning è diventato uno dei principali strumenti utilizzati in ambito informatico grazie alla grande mole di dati che è possibile ormai raccogliere e si può addirittura pensare che il suo influsso negli prossimi anni si accrescerà ulteriormente [SV08].

In questo ambito ricoprono grande importanza le reti neurali, un tecnologia che che si è rivelata molto adatta all'individuazione ed al riconoscimento di pattern statistici [Bis06], in particolare i *multilayer perceptron*, i quali si basano sulla combinazione di una serie di funzioni con pesi variabili.

In ciascuno dei nodi del primo livello ad esempio (come mostrato in 2.6) viene calcolato un certo valore di attivazione pari a

$$a_j = \sum_{i=1}^{D} w_{ij}^{(1)} x_i + w_{j0}^{(1)} \tag{2.4}$$

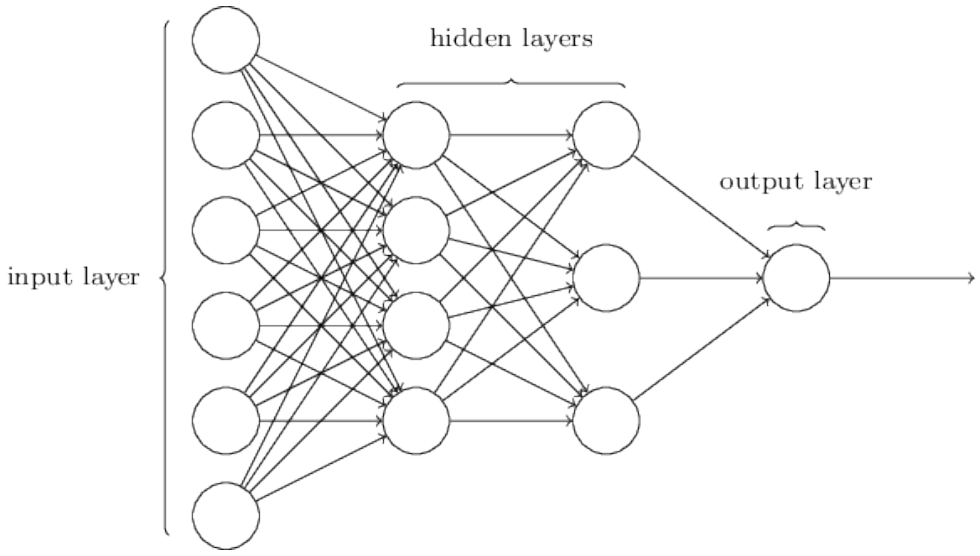

Figure 2.6. Un esempio di rete neurale *multilayer perceptron* in cui è possibile distinguere diversi livelli di nodi: un livello di input, due layer nascosti ed uno di output

con D numero di ingressi della rete, x_i valore dell'input i-esimo; i $w_{j0}^{(1)}$, chiamati *bias*, sono coefficienti fissati del primo layer e $w_{ij}^{(1)}$, chiamati *weights*, coefficienti che verranno modificati per adattarsi al modello che dovranno stimare.

L'output del nodo j-esimo (che verrà in seguito utilizzato come precedentemente era stato fatto con gli input) sarà quindi [Bis06]

$$z_j = h(a_j) \tag{2.5}$$

con $h()$ funzione non lineare definita *activation function*, che solitamente corrisponde ad una *tanh*, ad una ReLU, definita come [Nwa+18]

$$ReLU(x) = \max(x, 0) \tag{2.6}$$

oppure ad una sigmoide [Nwa+18]

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

Nel layer finale questa funzione di attivazione varia a seconda della tipologia di problema: nel caso di regressione corrisponde all'identità, viceversa nella classificazione viene spesso usata una softmax [Kec14], che restituisce per l'elemento in posizione j nel layer di output

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.8}$$

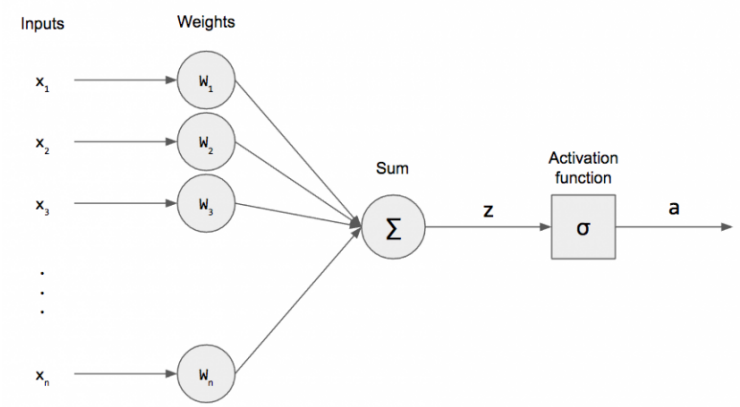

Figure 2.7. Le operazioni algebriche per il calcolo dell'output di un singolo nodo

Backpropagation

Fase fondamentale per una rete neurale è quella nella quale vengono analizzati i dati a disposizione (*training*): è questa la fase nella quale, attraverso la cosiddetta *backpropagation*, i pesi vengono adattati al problema in questione.

Questo corrisponde a cercare di minimizzare la seguente funzione di errore [Bis06]

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} ||\mathbf{y}(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n}||^2 \tag{2.9}$$

con $\mathbf{y}(\mathbf{x_n}, \mathbf{w})$ output della rete e $\mathbf{t_n}$ target.

Questo miglioramento della rete viene effettuato modificando i pesi delle funzioni precedentemente definite, propagando all'indietro nella rete neurale il valore dell'errore: viene calcolata la dipendenza dell'errore da ciascuno dei pesi, i.e.

$$\frac{\partial E}{\partial w_{ij}} \tag{2.10}$$

Che viene quindi utilizzato per modificare i pesi nel seguente modo [Maz15]

$$w'_{ij} = w_{ij} - \eta\frac{\partial E}{\partial w_{ij}} \tag{2.11}$$

con η *learning step*, un parametro definito inizialmente.

2.3 Programmazione lineare

La programmazione matematica studia la teoria ed i metodi utili per la ricerca di massimi o minimi di una funzione matematica, i.e. problemi del tipo [Wal85]

$$\text{minimizzare (o massimizzare) } f(x_1, \ldots, x_n) \text{ soggetta a } (x_1, \ldots, x_n) \text{ in } X$$

All'interno di essa ricopre grande importanza la programmazione lineare, che si occupa in particolare di risolvere problemi i cui vincoli e la funzione da minimizzare o massimizzare (detta *funzione obiettivo*) sono tutte relazioni lineari[Wal85]. Si può facilmente vedere come un generico problema di questo tipo, (ad esempio di minimizzazione) possa essere scritto nella forma[90]

$$\min f(x)$$
$$Ax \geq b, x \geq 0$$

2.3.1 Programmazione lineare multiobiettivo

Un'ulteriore classe di problemi di programmazione matematica è costituita dai cosidetti problemi multiobiettivo. In questo caso, denotando con

$$\theta = (\theta_1(x), \ldots, \theta_p(x))$$

il vettore delle p funzioni obiettivo, allora il problema consiste nel trovare le cosiddette soluzioni efficienti (o **ottimi di Pareto**), ovvero tutte $\bar{x}$ tali che non esiste x ammissibile (cioè che rispetta tutti i vincoli) tale che $\theta(x) \leq \theta(\bar{x})$ (nel caso di minimo) e $x \neq \bar{x}$ [Wal85].

L'insieme di queste soluzioni viene solitamente definito **frontiera di Pareto**, ed essa è graficamente visibile se si rappresenta l'insieme dei valori ammissibili nello spazio degli obiettivi Y, cioè l'insieme dei valori assunti da tutte $x \in X$ (le x ammissibili).

Nel caso di due funzioni obiettivo (Figure 2.8) i punti appartenenti alla frontiera di Pareto sono facilmente verificabili attraverso la regola del quadrante inferiore, cioè non vi devono essere elementi contenuti nel quadrante in basso a sinistra rispetto al punto considerato.

Teorema di Geoffrion

Per il calcolo delle soluzioni efficienti si rivela molto utile il teorema di Geoffrion, il quale enuncia che [FEG06], considerando il seguente problema multiobiettivo

$$\min(Cx)$$
$$Ax \geq b, x \geq 0 \tag{2.12}$$

con $C \in \mathbb{R}_{p \times n}$ matrice delle p funzioni obiettivo,

allora una soluzione x è un ottimo di Pareto $\iff$ $\exists \lambda = (\lambda_1, \ldots, \lambda_p) \geq 0$, $\sum_{i=1}^{p} \lambda_i = 1$ tale che x è ottima per il seguente problema

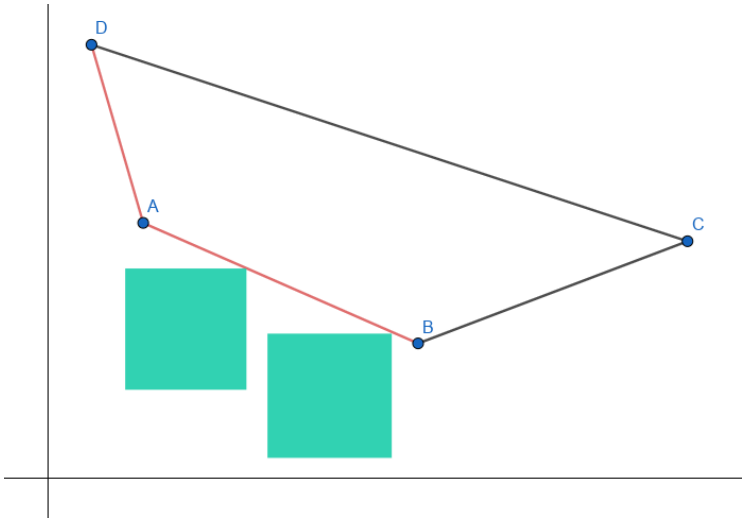

Figure 2.8. I segmenti in rosso rappresentano la frontiera di Pareto in un ipotetico problema di minimo: i punti da cui sono generati i quadranti inferiori non contengono altri elementi dell'insieme ammissibile nello spazio delle soluzioni

$$\min(\lambda C x)$$
$$Ax \geq b, x \geq 0$$

2.3.2 Programmazione lineare intera multiobiettivo

Nel caso in cui il problema multiobiettivo ha l'ulteriore vincoli delle variabili intere, allora per il calcolo della frontiera di Pareto (trasformando il problema in un singolo obiettivo, processo detto di *scalarizzazione*) si può ricorrere a due diversi metodi: il **metodo dei pesi** oppure il metodo **ϵ-constraints**.

Metodo dei pesi

Il metodo dei pesi è il più noto metodo di scalarizzazione: esso consiste nel far variare $\lambda = (\lambda_1, \ldots, \lambda_p) \geq 0$, con $\sum_{i=1}^{p} \lambda_i = 1$ trasformando 2.12 (dotato dell'ulteriore vincolo $x \in \mathbb{Z}^n$) in

$$\min(\lambda C x)$$
$$Ax \geq b, x \geq 0$$

E' importante però notare che questo metodo non permette di calcolare tutta la frontiera di Pareto: esso genere solo un sottoinsieme di punti che si definisce **supportato**.

Metodo ϵ-constraints

In questo caso una sola delle p funzioni obiettivo rimane tale mentre le altre sono trasformate in dei vincoli [FEG06].

Il problema è quindi riformulato nella forma (per $x \in \mathbb{Z}^n$)

$$\min(c_k x)$$
$$\min(c_i x \leq \epsilon_i) \quad \forall i \in \{1, \ldots, p\}, i \neq k$$
$$Ax \geq 0, x \geq 0$$

A differenza del metodo dei pesi, riesce a generare tutte le soluzioni efficienti.

2.4 Lavori precedenti

Le problematiche che si pone di risolvere questa applicazione sono state precedentemente trattate in altri lavori: l'idea di progettare un sistema completo e automatico stfruttando la face recognition e/o detection da utilizzare in ambito accademico era già stata proposta in "Face Recognition-based Lecture Attendance System", "On the Technologies and Systems for Student Attendance Tracking", *Student Attendance Recording System Using Face Recognition with GSM Based* e *Effective Mechanism for Advancement of monitoring Process of Educational Sectors of Underdeveloped Countries – A Study Based on Educational Sector of Pakistan.*

Ugualmente ha attirato diverse teorie ed applicazioni il tentativo di costruire un modello che potesse stimare il flusso di studenti, ad esempio in *An undergraduate student flow model: Australian higher education*, "Class Attendance in Undergraduate Courses" e "Students' Motivations for Class Attendance", nel quale in particolare, attraverso diverse ricerche, si cerca di individuare e quantificare il legame tra presenza alle lezioni e fattori come la materia stessa o il momento in cui essa si svolge.

Diversi articoli sono poi stati pubblicati riguardo il problema di allocazione delle aule tra le diverse classi e la sua formulazione matematica, come **Ola19** "Allocation of Classrooms by Linear Programming", "Solving the Course - Classroom Assignment Problem for a University", "When Is the Classroom Assignment Problem Hard?" e "Integer programming methods for large-scale practical classroom assignment problems".

Chapter 3

Metodi

Per lo sviluppo di questa applicazione si è scelto il linguaggio Python per la sua semplicità e immediatezza; ugualmente si sarebbe potuto optare per linguaggi come C++ o Java senza in alcun modo dover modificare la logica sviluppata.

3.1 Individuazione degli studenti

Il primo *step* che è necessario affrontare nello sviluppo di questa applicazione è la *face detection* per individuare il numero di studenti presenti nelle immagini fornite.

Si è deciso di utilizzare, per ovviare a questa problematica, alla libreria offerta da OpenCV, la quale offre inoltre dei classificatori pre-trainati (che utilizzano gli algoritmi introdotti in 2.1.3).

Tra questi sono stati scelti un modello per il riconoscimento dei volti visti frontalmente ed uno per il riconoscimento di quelli visti di profilo, rispettivamente il `haarcascade_frontalface_default.xml` e `haarcascade_profileface.xml`.

```
face_cascade_frontal =
 ↪  cv.CascadeClassifier(os.path.join(CLASSIFIERS_FOLDER,
 ↪  CLASSIFIER_FILENAME_FRONTAL))
face_cascade_profile =
 ↪  cv.CascadeClassifier(os.path.join(CLASSIFIERS_FOLDER,
 ↪  CLASSIFIER_FILENAME_PROFILE))
```

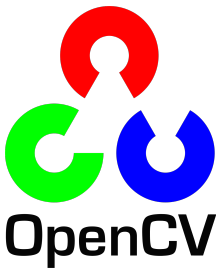

Figure 3.1. Il logo di OpenCV, una libreria molto utilizzata per la *computer vision*

Le facce riconosciute sono uguali all'unione di quelle riconosciute dai singoli classificatori

```
# detect faces
faces_profile = face_cascade_profile.detectMultiScale(img_gray,
↪  1.3, 5)
faces_frontal = face_cascade_frontal.detectMultiScale(img_gray,
↪  1.3, 5)

# merge faces detected
faces = list(faces_profile) + list(faces_frontal)
```

Per evitare che delle facce, riconosciute da entrambi i modelli, siano contate più di una volta, è stato poi inserito un ulteriore controllo con una soglia sull'area di overlapping dei rettangoli che contengono i visi: in questo modo se l'area in comune tra due di essi rapportata all'area di uno dei due è maggiore di una certa soglia, uno dei due è scartato.

```
valid_faces = []
# filter overlapping faces
for i in range(len(faces)):
    (x1, y1, w1, h1) = faces[i]
    other_faces = faces[i + 1:]

    valid_faces.append(faces[i])

    for (x2, y2, w2, h2) in other_faces:
        overlapping_area = get_overlapping_area(x1, y1, w1, h1, x2,
        ↪  y2, w2, h2)

        if overlapping_area / w1 * h1 > OVERLAPPING_THRESHOLD:
            valid_faces.pop()
            break
```

avendo precedentemente definito

```
OVERLAPPING_THRESHOLD = 0.9
```

che, in seguito a diversi test, si è rivelato un valore valido per il rilevamento dei rettangoli che si sovrapponevano, e

```
def get_overlapping_area(x1, y1, w1, h1, x2, y2, w2, h2):
    """
    Calculates the overlapping area given the two rectangles
↪   respectively defined by (x1, y1, w1, h1)
    and (x2, y2, w2, h2)
    """
```

3.2 Il modello di regressione

3.2.1 Struttura della rete

Per stimare il numero di studenti viene utilizzato un modello di rete neurale molto semplice: essa consiste in un layer di input, due layer nascosti, con funzione di attivazione la ReLU 2.6, ed un layer di output. Per l'implementazione si utilizza Keras, una libreria che offre un interfaccia immediata e completa per il machine learning in Python.

```
# define base model
def baseline_model(inputs=9):
    # create model
    model = Sequential()
    model.add(Dense(inputs, input_dim=inputs,
    ↪  kernel_initializer='normal', activation='relu'))
    model.add(Dense(12, kernel_initializer='normal',
    ↪  activation='relu'))
    model.add(Dense(12, kernel_initializer='normal',
    ↪  activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))

    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam',
    ↪  metrics=[r_square, rmse])

    return model
```

Come funzione di loss il modello utilizzerà l'MSE (Mean Square Error, o errore quadratico medio) definito come (essendo e_j lo scostamento nella stima j-esima di n)[Bot18]

$$MSE = \frac{1}{n}\sum_{j=1}^{n} e_j^2 \tag{3.1}$$

e come metriche ulteriori il RMSE (Root Mean Square Error), che corrisponde alla radice quadrata di 3.1 e R *squared*, calcolato come [Col18]

$$R^2 = 1 - \frac{SSE}{SST} \tag{3.2}$$

con SSE somma dei quadrati degli errori e SST somma dei quadrati degli scostamenti dei valori reali dalla media.

Di particolare interesse è la prima metrica, in quanto risulta indicatrice del numero di studenti di differenza tra la stima effettuata ed i dati analizzati.

```
def rmse(y_true, y_pred):
    from keras import backend
```

```
    return backend.sqrt(backend.mean(backend.square(y_pred -
    ↪  y_true), axis=-1))

def r_square(y_true, y_pred):
    from keras import backend as K
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

3.2.2 Training

Il dataset utilizzato per il training (generato artificialmente) consiste in una serie di entry del tipo

```
{"hour": 18, "students": 100, "day": "thursday", "subject": "CAL"}
```

(sono stati inseriti anche il numero di studenti poichè generare un dataset con delle label associate a foto sarebbe stato molto dispendioso e non necessario, visto che l'implementazione scelta dell'applicazione permette sia di partire da foto con informazioni associate sia direttamente da entry come quella appena mostrata, essendo la parte di face detection separata da quella di machine learning).

La vera fase di training quindi viene effettuata nella seguente funzione, definita per la classe `StudentsEstimator`, un wrapper per il modello di regressione e che effettua anche la fase di encoding dei dati

```
def train(self, early_stopping=True):
    # train model
    callbacks = []

    if early_stopping:
        early_stopping = EarlyStopping(monitor='val_loss',
        ↪  patience=10)
        callbacks.append(early_stopping)

    history = self.estimator.fit(self.X, self.Y,
    ↪  validation_split=VALIDATION_FRACTION, callbacks=callbacks)
```

con `VALIDATION_FRACTION` precedentemente definita pari a 0.1.

Se non specificato, viene effettuato l'*early stopping*, cioè il training viene interrotto nel momento in cui per un numero di epoche consecutive pari alla `patience` la loss nei test cresce anziché diminuire (questo procedimento viene solitamente aggiunto per evitare *overfitting* sui dati, che cioè la rete diventi un modello adatto a rappresentare ottimamente solo i dati di training e performi molto male sui restanti).

Gli score e le metriche precedentemente introdotti verranno poi estrapolati dalla `history` restituita da `estimator.fit()`.

A questo punto vengono effettuate una o più stime attraverso il modello costruito, dopo aver codificato i dati in modo coerente rispetto al training

Figure 3.2. Keras è una API ad alto livello per il machine learning in Python e che si può basare su TensorFlow, CNTK, e Theano

```
def predict(self, prediction_data):
    # encode data before prediction
    dp = pandas.DataFrame(prediction_data)
    X = self.transform_data(dp)

    predictions_float = self.estimator.predict(X)
```

Si dispone quindi di entry identiche nella forma a quelle utilizzate per il training, con la differenza che le prime sono ottenute attraverso delle stime. Si passa quindi all'ultima parte dell'applicazione, incentrata sull'allocazione dei gruppi di studenti.

Bibliography

[90] *Introduction to Linear Programming: Applications and Extensions (Chapman & Hall/CRC Pure and Applied Mathematics)*. CRC Press, 1990. ISBN: 978-0824783839. URL: https://www.amazon.com/Introduction-Linear-Programming-Applications-Mathematics/dp/0824783832?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0824783832.

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[Ble92] Malcolm L. Van Blerkom. "Class Attendance in Undergraduate Courses". In: *The Journal of Psychology* 126.5 (Sept. 1992), pp. 487–494. DOI: 10.1080/00223980.1992.10543382.

[Bot18] Alexei Botchkarev. "Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology". In: *Interdisciplinary Journal of Information, Knowledge, and Management, 2019, 14, 45-79* (Sept. 9, 2018). DOI: 10.28945/4184. arXiv: http://arxiv.org/abs/1809.03006v1 [stat.ME].

[CC08] Dan Campbell and Sherlock Campbell. "Introduction to regression and data analysis". In: *StatLab Workshop Series*. 2008, pp. 1–15.

[Col18] Thomas Colignatus. *An overview of the elementary statistics of correlation, R-squared, cosine, sine, and regression through the origin, with application to votes and seats for Parliament*. MPRA Paper. University Library of Munich, Germany, 2018. URL: https://EconPapers.repec.org/RePEc:pra:mprapa:84722.

[CT92] Michael W. Carter and Craig A. Tovey. "When Is the Classroom Assignment Problem Hard?" In: *Operations Research* 40.1-supplement-1 (Feb. 1992), S28–S39. DOI: 10.1287/opre.40.1.s28.

[DDB15] Asit Kumar Datta, Madhura Datta, and Pradipta Kumar Banerjee. *Face Detection and Recognition*. Taylor & Francis, Oct. 28, 2015. 352 pp. URL: https://www.ebook.de/de/product/25204083/asit_kumar_datta_madhura_datta_pradipta_kumar_banerjee_face_detection_and_recognition.html.

[FEG06] JosÉ Figueira, Matthias Ehrogott, and Salvatore Greco. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer-Verlag GmbH, Jan. 20, 2006. URL: https://www.ebook.de/de/product/11429977/jose_figueira_matthias_ehrogott_salvatore_greco_multiple_criteria_decision_analysis_state_of_the_art_surveys.html.

[Fjo05] Nancy Fjortoft. "Students' Motivations for Class Attendance". In: *American Journal of Pharmaceutical Education* 69.1 (Sept. 2005), p. 15. DOI: 10.5688/aj690115.

[GT86] Karl Gosselin and Michel Truchon. "Allocation of Classrooms by Linear Programming". In: *Journal of the Operational Research Society* 37.6 (June 1986), pp. 561–569. DOI: 10.1057/jors.1986.98.

[Kan14] Kanjana Thongsanit. "Solving the Course - Classroom Assignment Problem for a University". In: *Silpakorn University Science and Technology Journal-1* 8 (2014). DOI: 10.14456/sustj.2014.3.

[Kaw+05] Yohei Kawaguchi et al. "Face Recognition-based Lecture Attendance System". 2005. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.560.7003&rep=rep1&type=pdf.

[Kec14] Christian Keck. *Models for correspondence finding and probabilistic representative learning*. epubli GmbH, 2014. ISBN: 978-3-8442-9787-4. URL: https://www.amazon.com/correspondence-finding-probabilistic-representative-learning/dp/3844297871?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3844297871.

[Kot+18] Zoran Kotevski et al. "On the Technologies and Systems for Student Attendance Tracking". In: *International Journal of Information Technology and Computer Science* 10.10 (Oct. 2018), pp. 44–52. DOI: 10.5815/ijitcs.2018.10.06.

[Maz15] Matt Mazur. *A Step by Step Backpropagation Example*. 2015. URL: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/.

[Nwa+18] Chigozie Nwankpa et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: (Nov. 8, 2018). arXiv: http://arxiv.org/abs/1811.03378v1 [cs.LG].

[Phi+15] Antony E. Phillips et al. "Integer programming methods for large-scale practical classroom assignment problems". In: *Computers & Operations Research* 53 (Jan. 2015), pp. 42–53. DOI: 10.1016/j.cor.2014.07.012.

[PKJ14] Mr. C. S. Patil, Mr. R. R. Karhe, and Mr. M. D. Jain. *Student Attendance Recording System Using Face Recognition with GSM Based*. Ed. by International Journal of Research in Advent Technology. 2014. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.640.4151&rep=rep1&type=pdf.

[SB99] Chandra Shah and Gerald Burke. *An undergraduate student flow model: Australian higher education*. Ed. by Kluwer Academic Publishers. 1999. URL: https://s3.amazonaws.com/academia.edu.documents/46873010/a_3A100376522225020160628-24881-76g4vs.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1559310657&Signature=VmZdxJEzPWnXXBX3UJXI15NVkjo%3D&response-content-disposition=inline%3B%20filename%3DAn_undergraduate_student_flow_model_Aust.pdf.

[Sch13] Robert E. Schapire. "Explaining AdaBoost". In: *Empirical Inference*. Springer Berlin Heidelberg, 2013, pp. 37–52. DOI: 10.1007/978-3-642-41136-6_5.

[SS16] Muhammad Zulqarnain Siddiqui and Prof. Dr. P. Sellappan. *Effective Mechanism for Advancement of monitoring Process of Educational Sectors of Underdeveloped Countries – A Study Based on Educational Sector of Pakistan*. Ed. by International Journal of Scientific & Engineering Research. 2016. URL: https://www.researchgate.net/profile/Muhammad_Siddiqui65/publication/315689570_Effective_Mechanism_for_Advancement_of_Monitoring_Process_of_Educational_Sectors_of_Underdeveloped_Countries_-_A_Study_Based_on_Educational_Sector_of_Pakistan/links/58dbabc392851c611d0091f1/Effective-Mechanism-for-Advancement-of-Monitoring-Process-of-Educational-Sectors-of-Underdeveloped-Countries-A-Study-Based-on-Educational-Sector-of-Pakistan.pdf.

[SV08] Alex Smola and S. V. N. Vishwanathan. *Introduction to Machine Learning*. Ed. by Cambridge University Press. 2008. URL: http://alex.smola.org/drafts/thebook.pdf.

[VJ04] Paul Viola and Michael J. Jones. "Robust Real-Time Face Detection". In: *International Journal of Computer Vision* 57.2 (May 2004), pp. 137–154. DOI: 10.1023/b:visi.0000013087.49260.fb.

[Wal85] Gordon Raymond Walsh. *An introduction to linear programming*. Wiley New York, 1985.

[Wan14] Yi-Qing Wang. "An Analysis of the Viola-Jones Face Detection Algorithm". In: *Image Processing On Line* 4 (June 2014), pp. 128–148. DOI: 10.5201/ipol.2014.104.