Università degli Studi di Padova

Dipartimento di Matematica
Corso di Laurea in Informatica

# Classificazione firme statiche utilizzando i Hidden Markov Models

*Relatore:*
Ch.mo Prof. Tullio Vardanega

*Laureando:*
Alexandru Prigoreanu
1004887

Anno accademico 2012/2013

**Sommario**

La presente relazione ha come scopo la descrizione dell'attività di stage, svolta dal sottoscritto, nel periodo settembre-ottobre 2013 presso l'azienda Corvallis. Il primo capitolo descrive l'azienda ospitante. Il secondo capitolo espone le motivazioni e gli obiettivi del progetto di stage. Il terzo capitolo illustra in modo approfondito le attività effettuate per raggiungere gli obiettivi prefissati. Il quarto ed ultimo capitolo riporta una valutazione a posteriori sul lavoro svolto, sulle conoscenze acquisite e sulla distanza tra le conoscenze richieste e le conoscenze possedute.

## Convenzioni tipografiche

Al fine di migliorare la leggibilità e la chiarezza dei contenuti esposti sono state adottate le seguenti convenzioni tipografiche:

- Problema : pdf ci sono i link andata e ritorno ... ma stampato no!, definisci convenzioni che valgono per entrambe le situazioni, per la legge dell'accessibilità generale il colore non deve veicolare informazioni
- *corsivo* per i termini in lingua diversa dall'italiano
- *corsivo* per i nomi propri che individuano tecnologie

# Indice

# Elenco delle tabelle



# Elenco delle figure

# 1 Dominio applicativo

La prima parte di questa relazione si prefigge di presentare al lettore il contesto di lavoro dell'azienda ospitante il progetto di stage. Inizialmente descrivo brevemente l'azienda. A seguire effettuo una panoramica sui prodotti e servizi che essa offre e sui clienti che cerca di soddisfare. Infine elenco alcune delle tecnologie di appoggio allo sviluppo software e i processi interni di quest'ultimo.

## 1.1 Azienda


Figura 1: Logo azienda Corvallis

Corvallis S.p.A. (http://www.corvallis.it/) è una società italiana presente nel settore dell'*Information Technology*. Essa applica le sue competenze funzionali, tecnologiche e di processo, acquisite in quasi trenta anni di operatività, in 4 settori strategici per il mondo finance:

- risparmio gestito e crediti;
- *document management*;
- area compliance;
- evoluzione tecnologica.

Corvallis fornisce prodotti e servizi a clienti appartenenti all'ambito finanziario (banche e assicurazioni), industria/servizi e Pubblica Amministrazione. In particolare, Corvallis effettua attività di progettazione e sviluppo software, di fornitura di prodotti software propri e di terzi, di manutenzione e assistenza tecnica, di consulenza.
Il Gruppo Corvallis è composto da più di 600 risorse, tra dipendenti e collaboratori, distribuite in 13 sedi operative nel territorio italiano.

## 1.2 Prodotti e clienti

### 1.2.1 Banche

Corvallis opera da molti anni con i principali istituti di credito e società prodotto. Le aree di competenza sviluppate riguardano:

- Risparmio Gestito;
- Finanza;
- Crediti;
- Compliance;

- *Governance*;
- Sistemi di Pagamento;
- *Document Management*.

### 1.2.2 Assicurazioni

Corvallis può effettuare *outsourcing* nelle compagnie Assicurative per i processi Vita e Danni (IT e *back office*). L'offerta verso le compagnie assicuratrici riguarda le seguenti aree:

- Sistema Portafoglio Vita e Danni;
- Sistemi di agenzia;
- Sistemi sinistri;
- Compliance;
- *Governance*;
- Servizi di *Back Office*;
- *Outsourcing*;
- *Document Management*.

### 1.2.3 Industria e servizi

I clienti di Corvallis sono alcune industrie e imprese appartenenti a differenti settori: *Oil & Gas*, *Engineering*, *Construction*, *Aerospace & Defense*, Telco, Media. Corvallis possiede competenze anche nell'ambito del *Project & Portfolio Management*. Inoltre svolge erogazione di servizi di supporto specialistico relativo a prodotti leader di mercato tra i quali Cognos, Qlikview, Hyperion, Business Object e IrionDQ.

### 1.2.4 Pubblica Amministrazione

Corvallis offre soluzioni e servizi alle pubbliche amministrazioni. Alcune di queste riguardano:

- Tributi e fiscalità per gli enti locali;
- Sistemi informativi territoriali e catastali, *GIS*, *Digital Mapping* e cartografia;
- *Document Management*;
- Sistemi Informativi per la catalogazione, conservazione e la tutela dei Beni Culturali.

### 1.2.5 *Know Your Customer* (*KYC*)

*Know Your Customer* è un prodotto sviluppato dall'azienda Corvallis per soddisfare i requisiti normativi richiesti dalla direttiva Cee in materia di Antiriciclaggio. É un software utile alle banche e alle assicurazioni poiché permette di tenere sotto controllo la segnalazione di operazioni sospette previste dalla normativa. Il prodotto si compone di sette moduli integrabili e flessibili:

- *Repository*, modulo che archivia in un unico ambiente tutte le informazioni relative al cliente e la sua operatività, necessarie a soddisfare le diverse richieste imposte dalla normativa;
- *Classifier*, modulo che racchiude una generale profilatura del Cliente rispetto ad un numero non prefissato di dimensioni, indipendenti tra loro ma con la condivisione di dati e funzioni;
- *Integrator*, modulo di integrazione delle fonti esterne, fornisce tutte le funzioni richieste per l'import e l'aggregazione delle liste sia pubbliche che private (ad esempio Persone Politicamente Esposte);
- *Expansion*, modulo basato su logiche di intelligenza artificiale, soddisfa l'esigenza di ricerche di ricorrenze di dati, presenti nelle liste pubbliche, nei database interni, nei database forniti da *info providers*, oppure nei dati acquisiti tramite il modulo *Integrator*;
- *Control*, modulo che presidia l'analisi dell'operatività della clientela considerando i dati anagrafici, di rapporto e di movimentazione, generando degli aggregati statistici da analizzare;
- Monitor, modulo che propone un'interfaccia utente per visualizzare e analizzare gli *alert* generati dai moduli *Classifier*, *Expansion* e *Control*;
- *Configurator*, modulo che permette di assicurare un approccio flessibile ed efficiente che può essere facilmente modificato rispondendo ai cambiamenti repentini delle regole internazionali o interne all'organizzazione. Il modulo governa le parametrizzazioni delle altre componenti del prodotto *KYC*.

Il software è stato scritto in *Java* utilizzando il *framework* aziendale.

## 1.3 Tecnologie

Seguono alcune delle tecnologie impiegate nello sviluppo software.
I linguaggi di programmazione più utilizzati sono *Java* e *Cobol*.
Il sistema di versionamento in uso è il *Concurrent Versions System* (*CVS*).
Alcuni dei database utilizzati sono: *DB2*, *Oracle*, *MS SQL*, *PostgreSQL*.
I *framework* maggiormente utilizzati sono *JBoss* e *JWolf*. *JWolf* è stato creato dall'azienda stessa.É un *framework* per lo sviluppo di applicazioni su tecnologia *Java* multipiattaforma e multicanale e racchiude le *best practices* dell'architettura Service Oriented Architecture (SOA).
L'ambiente di sviluppo per *Java* è *Eclipse*.

In figura 2 possiamo vedere uno schema riassuntivo di alcune delle tecnologie in uso.

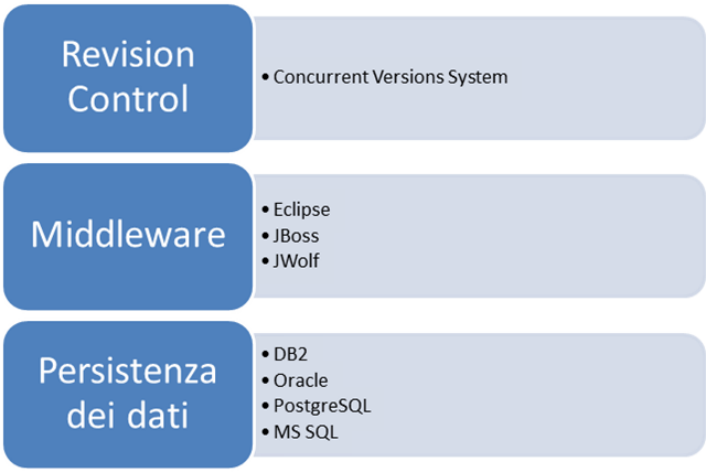



Figura 2: Tecnologie in uso

## 1.4 Processi interni

Le fasi di sviluppo di un progetto sono costituite da:

- Coordinamento e Riunioni. In questa fase vengono pianificate tutte le attività necessarie allo svolgimento del progetto. Gli incontri con i clienti hanno come scopo una efficiente trasmissione di informazioni;
- Analisi dei requisiti. L'output dell'attività di analisi è un documento in cui vengono racchiusi tutti i requisiti funzionali, qualitativi, prestazionali e dichiarativi dei quali il prodotto finale dovrà garantirne il soddisfacimento. Il documento serve da input per la fase di progettazione;
- Progettazione. Nella fase di progettazione si definiscono le specifiche tecniche delle funzionalità da realizzare. Il risultato di questa fase è il documento di Specifiche Tecniche di Progettazione;
- Sviluppo. É lo stadio esecutivo del progetto con il quale si realizzano i moduli software previsti dal disegno applicativo. In questa fase vengono effettuati test di unità;
- Test funzionali e di sistema. I test funzionali hanno lo scopo di verificare che i moduli realizzati durante la fase di codifica rispettino quanto fissato dai requisiti iniziali. Il test di sistema valida il prodotto nella sua interezza;
- Collaudo con il cliente. Si tratta di un test di sistema effettuato su un ambiente del cliente e con dati di prova forniti dallo stesso. L'output di questa attività è un verbale che racconta l'esito del collaudo;
- Documentazione di prodotto. Questa fase prevede la stesura dei Manuali di Prodotto relativi al software realizzato;

L'output di ogni fase viene verificato e, se conforme agli standard di qualità dell'azienda, approvato. Altrimenti dovranno essere indicate delle misure correttive per i problemi individuati.

In figura 3 vediamo un resoconto delle fasi necessarie allo sviluppo software.


Figura 3: Sviluppo Software, ciclo di vita

# 2 Progetto aziendale

Questo capitolo ha lo scopo di mostrare al lettore l'ambito in cui si colloca il progetto di stage proposto dall'azienda Corvallis. Inizialmente presento i motivi che hanno portato alla nascita del progetto. A seguire effettuo una panoramica sulla classificazione delle firme statiche. In seguito descrivo il prototipo/classificatore preesistente al progetto di stage. Infine illustro gli obiettivi, le aspettative e i vincoli del progetto aziendale a cui ho preso parte.

Visti i notevoli vantaggi in termini di incremento dell'efficienza e di riduzione dei costi che la dematerializzazione garantisce (risparmio relativo ai costi di stampa, acquisto e manutenzione delle stampanti), nell'ambito delle nuove normative alle banche sarà concesso lo scambio di immagini degli assegni bancari.
In base a questa premessa l'azienda Corvallis ha intuito che il core di un'applicazione bancaria che accetta lo scambio di immagini degli assegni bancari sarà un classificatore di firme statiche accurato, robusto e affidabile. Conseguentemente ha implementato un prototipo per la classificazione di firme statiche.

## 2.1 Introduzione alla classificazione di firme statiche

La firma è un tratto comportamentale di un individuo e costituisce una particolare classe di scrittura dove lettere o parole possono essere non distinguibili. É considerata un elemento distintivo avente caratteristiche uniche e personali. Accertare in maniera chiara ed univoca il sottoscrittore di un documento avente forza legale (in questo caso assegni bancari) è di fondamentale importanza. Infatti il destinatario deve poter identificare l'identità del mittente (autenticità) e il mittente non deve poter disconoscere un documento da lui firmato (non ripudio). Nasce quindi l'esigenza di distinguere tra firme false e firme autentiche.

Le difficoltà principali nel classificare le firme autografe sono dovute alle variazioni intrapersonali: le firme di una persona possiedono grande variabilità, dovuta allo stato emotivo dei sottoscrittori oppure alla posizione di raccolta, e di conseguenza, se confrontassimo due esemplari di firma di un firmatario questi non sarebbero identici. Invece l'agevolazione cardinale sta nel fatto che le firme di persone diverse manifestano caratteristiche elementari distinte.

A seconda del hardware front-end, un sistema di verifica della firma (*signature verification system*) può essere etichettato come *offline* o *online*. Nei sistemi *offline* la verifica della firma avviene dopo la sottoscrizione della firma. Le uniche informazioni che si possiedono sono di natura statica: l'immagine della firma autografa del sottoscrivente. Al contrario, nei sistemi online, le firme vengono acquisite tramite un dispositivo elettronico (tavoletta grafica) oppure con una penna speciale, capace di memorizzare una sequenza di punti che descrivono velocità, pressione, ritmo, accelerazione e movimento effettuati dal sottoscrittore, non solo l'immagine statica.

### 2.1.1 Valutazione delle performance

La performance di un sistema di verifica della firma si calcola in base alla percentuale di errori che esso commette nel classificare le firme. Esistono due tipi di errori. Questi vengono riassunti dai seguenti indici:

- *False Acceptance Rate* ($FAR$), indice di accettazione dei falsi, ossia percentuale delle firme false classificate come genuine;

$$FAR = \frac{nr.\ falsi\ accettati}{nr.\ falsi\ totali}$$

- *False Rejection Rate* (*FRR*), indice di rifiuto dei genuini, ossia percentuale delle firme genuine classificate come false.

$$FRR = \frac{nr.\ genuini\ rifiutati}{nr.\ genuini\ totali}$$

Purtroppo i due indici sono inversamente proporzionali: a una diminuzione del *FAR* corrisponde un aumento del *FRR* e viceversa a una diminuzione del *FRR* corrisponde un aumento del *FAR*. É evidente quindi che si deve arrivare a un compromesso: minimizzare il *FAR* conservando un valore tollerabile per il *FRR*.
Se si eguagliano i due indici si è in presenza di un nuovo indice: *Equal Error Rate* (*EER*). L'*EER* permette di paragonare in modo veloce l'accuratezza dei sistemi di verifica. In generale, più l'indice *EER* è basso più il sistema è accurato.
Non sorprendentemente, grazie ai dati biometrici che i sistemi di verifica della firma online possiedono in più, essi sono più accurati e affidabili dei sistemi di verifica della firma statica.

### 2.1.2 Tipi di falsificazione

In letteratura sono stati individuati tre tipi di falsificazione a seconda del grado di preparazione del falsificatore sulla firma che sta cercando di riprodurre:

- falsificazioni casuali (*random forgeries*), prodotte senza conoscere né il nome del firmatario né la forma della sua firma;
- falsificazioni semplici (*simple forgeries*), prodotte conoscendo il nome del firmatario ma senza avere un esempio della sua firma;
- falsificazioni accurate (*skilled forgeries*), prodotte dopo un allenamento con l'obiettivo di imitare la firma originale nel miglior modo possibile.

In figura 4 vediamo un esempio di firma genuina e degli esempi di falsificazione della stessa.


Figura 4: Tipi di falsificazioni:
(a) firma genuina; (b) falsificazione casuale;
(c) falsificazione semplice; (d) falsificazione accurata

### 2.1.3 Verifica della firma statica (*offline signature verification*)

La verifica di firme statiche appartiene alla classe di problemi del *pattern recognition* e un tipico sistema di *pattern recognition* si compone dai seguenti step [1]:

1. Acquisizione dati, cattura delle immagini firma;
2. *Preprocessing*, per semplificare le operazioni successive minimizzando la perdita di informazioni;
3. Estrazione delle caratteristiche (*feature extraction*), riduzione dei dati in input misurando solo determinate caratteristiche (*feature*) o proprietà;
4. Classificazione (oppure fase di verifica), prendere una decisione (accettare o rifiutare) sulla base dell'adeguatezza dei valori ottenuti dalla fase di *feature extraction*.

In figura 5 vediamo il *workflow* generale del processo di verifica della firma statica.

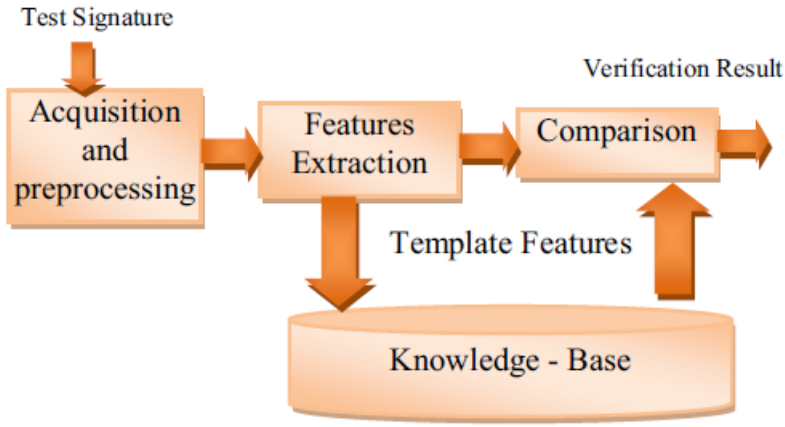



Figura 5: *Workflow* dei sistemi di verifica della firma statica
Sorgente: [4]

### Acquisizione dati

Le immagini di firma vengono scannerizzate oppure vengono estratte da documenti *PDF*.

### *Preprocessing*

Il *preprocessing* della firma è uno step fondamentale necessario a migliorare l'accuratezza dell'estrazione delle caratterisitche e della classificazione. Lo scopo della fase di *preprocessing* è quello di standardizzare le firme e renderle pronte per la fase di *feature extraction*.
Di seguito elenco alcune delle operazioni di *preprocessing* più utilizzate:

- Estrazione della firma (*signature extraction*);
- Ritaglio (*cropping*), l'immagine viene ridotta al rettangolo che circoscrive la firma;
- Ridimensionamento (*resizing*), l'immagine della firma viene ridimensionata;
- Binarizzazione (*binarization*), conversione da colore a scala di grigi e infine a bianco-nero (binario);
- Assottigliamento (*thinning*), assottigliamento del tratto della firma a un pixel per uniformare lo spessore;

Un'errata scelta degli algoritmi di *preprocessing* può comportare una considerevole perdita di informazione.

### Estrazione delle *features*

Il successo di un sistema di verifica della firma dipende fortemente dalla fase di Estrazione delle *features*. Un metodo ideale di *feature extraction* prevede l'estrazione di quelle caratteristiche che minimizzano le variazioni intrapersonali presenti nelle firme di un sottoscrittore e massimizzano le variazioni interpersonali presenti nelle firme di due sottoscrittori distinti[2].

A seconda della facilità con la quale si possono imitare, le *feature* possono essere classificate in *feature* statiche o *feature* pseudo-dinamiche. La prima categoria contiene caratteristiche percettive e quindi facili da imitare, mentre la seconda categoria contiene caratteristiche poco intuitive e quindi di difficile imitazione[3]. Segue un elenco stringato delle *features* più utilizzate in letteratura.
*Features* statiche:

- *Calibre*, valore che individua il rapporto tra altezza e larghezza della firma;
- *Proportion*, si riferisce alle simmetrie presenti nell'immagine
- *Spacing*, valore che individua il numero di blocchi che compongono la firma, se non vi sono spaziature questo valore è uguale a 1;
- *Base behavior*, descrive l'angolo di inclinazione della firma in base a un'immaginaria linea orizzontale.

*Features* pseudo-dinamiche:

- *Density of pixels*, descrive la larghezza dei tratti, viene considerata pressione apparente;
- *Distribution of pixels*, descrive il numero di pixel neri presenti in uno spazio;
- *Slant*, individua informazioni riguardo all'inclinazione del tratto in considerazione;
- *Form*, individua informazioni riguardo alla forma del tratto in considerazione;

Un'ulteriore suddivisione delle *features* è dovuta alla segmentazione. In presenza di un'immagine firma segmentata le *features* possono essere globali (riguardano l'immagine nella sua interezza) o locali (riguardano determinate parti dell'immagine).

In figura 6 vediamo la panoramica delle *features* sopra elencate.

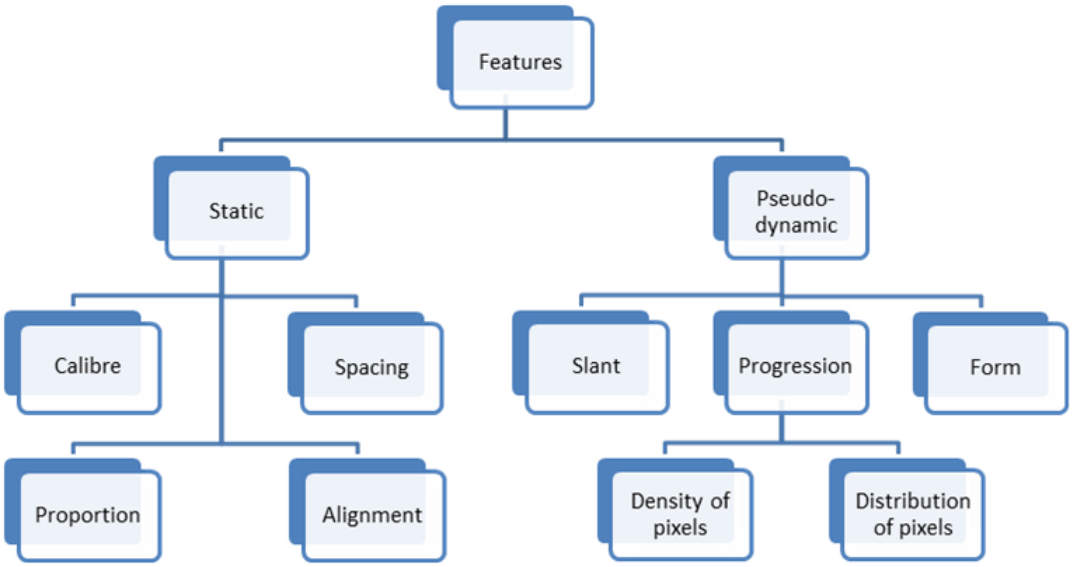



Figura 6: Features statiche e features pseudo-dinamiche

**Classificazione**

L'ultimo step nel processo di verifica dei sistemi *offline* decide se la firma presa in considerazione è genuina o falsa.
In letteratura esistono diversi tipi di approcci al problema della classificazione di firme statiche.

In figura 7 riporto un grafico riassuntivo degli approcci più utilizzati.

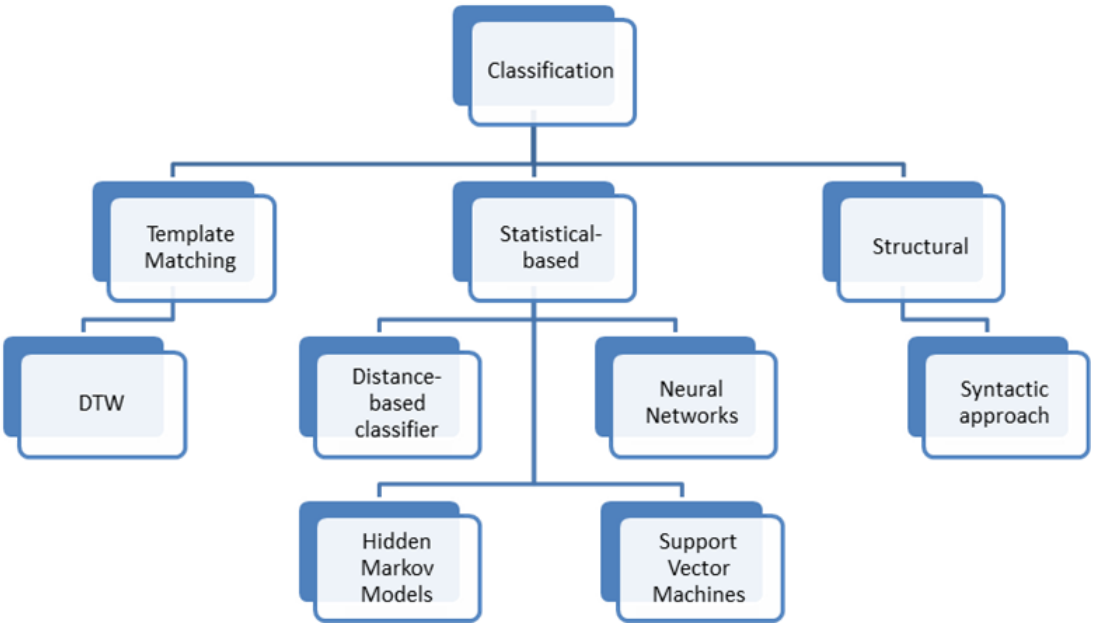



Figura 7: Panoramica metodi di classificazione

Poiché alcuni dei prossimi argomenti di questa relazione trattano i classificatori di tipo *Statistical-based*, né spiego la logica.

#### Classificatori *Statistical-based*

Nei classificatori di tipo *Statistical-based* la conoscenza statistica viene utilizzata per usufruire di nozioni come la relazione, la deviazione, tra due o più item per trovare qualche specifica correlazione tra essi. Nei sistemi di verifica della firma, la firma media (*template*/*pattern*) viene elaborata a partire da firme collezionate precedentemente (nella fase di training). Questa viene salvata nella base di conoscenza e quando si ha in input una firma da classificare viene utilizzato il concetto di correlazione per calcolare la distanza tra la firma da verificare e la firma media per poi decidere se accettarla o rifiutarla.

Nella prossima sezione descrivo il classificatore di firme statiche implementato dall'azienda Corvallis prima del progetto di stage.

### 2.1.4 Prototipo preesistente

Il classificatore di firme statiche, implementato dall'azienda, fa parte della categoria *Statistical-based*. In particolare è un classificatore di tipo *Distance-based*.

Il prototipo è in grado di estrarre 21 *features*. Il sistema viene addestrato utilizzando un database di firme. Per ciascun sottoscrittore, un vettore contenente il baricentro di ogni *feature* (*centroid feature vector*) viene calcolato utilizzando gli esemplari di firma genuini. Il vettore baricentro costituisce il *template* delle firme del firmatario. In fase di verifica, per misurare la distanza tra la firma *template* e la firma da testare, viene adoperata la distanza Euclidea nello spazio delle *features*.
La particolarità di questo sistema sta nel fatto che, per classificare una firma, non si basa soltanto sulla distanza Euclidea ma anche su un meccanismo dei pesi. Si assegna una peso ad ogni *feature* in

base alla variabilità intrapersonale che essa assume tra i prototipi firma di un sottoscrittore. Questa meccanica è descritta meglio nelle sezioni a seguire.

In figura 8 segue uno schema riassuntivo dei processi interni al classificatore.

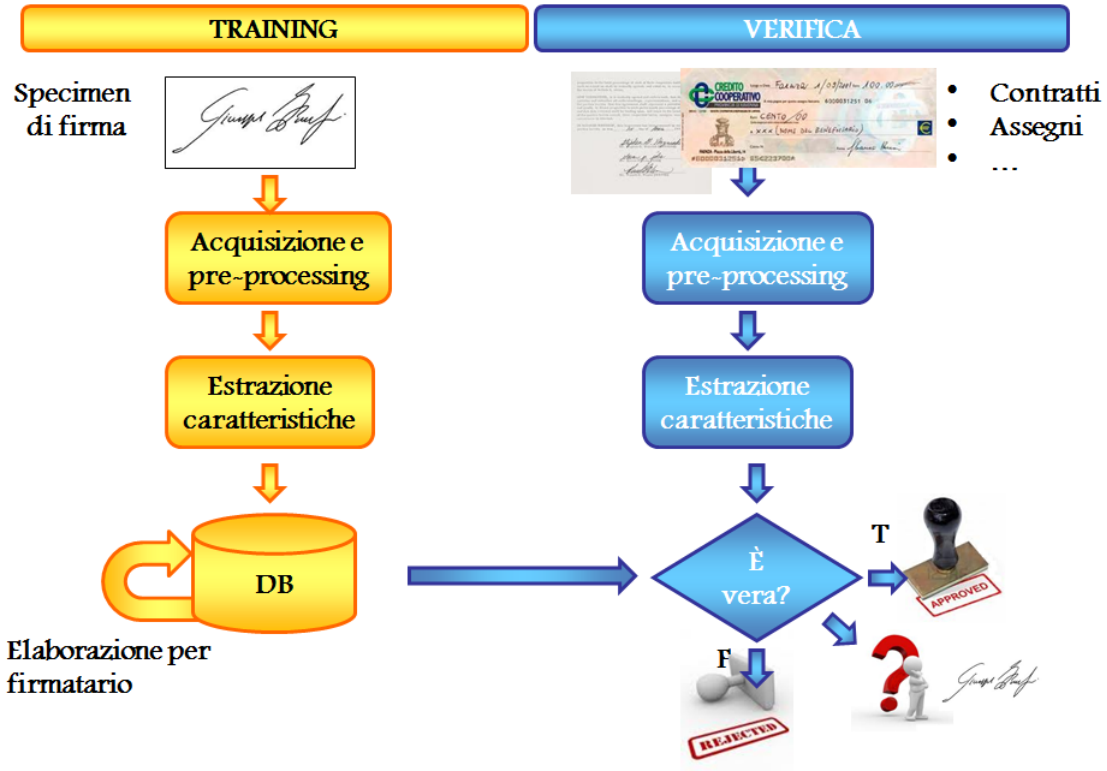



Figura 8: Il processo
Sorgente:Corvallis

### Fase di training

Il training prende in input N prototipi di firma genuina, una lista di *feature* da estrarre e i *preprocessing* necessari ad ogni *feature*. Restituisce in output un file contenente quattro attributi per *feature*, utili in fase di verifica:

- *average*, la media di una *feature* è data dalla somma dei valori che tale *feature* assume negli N prototipi diviso N:

$$Faverage = \frac{\sum_{i=1}^{N} FValPrototipoI}{N}$$

- *threshold*, la soglia di una *feature* è data dal massimo valore tra la massima distanza del valore della *feature* dalla media e il 10% della media:

$$Fthreshold = \max[\max(|FValPrototipoI - Faverage|), (Faverage * 0.1)]$$

- *weight*, il peso di una *feature* è diverso da utente a utente poiché è inversamente proporzionale alla variabilità della *feature* rispetto alle altre dell'utente considerato. Il peso di una *feature* appartiene all'intervallo [0, 1].
  - il peso vale 0 se la deviazione standard è maggiore del 30% della media;
  - il peso vale 1 se la deviazione standard è uguale a 0;
  - altrimenti:

$$Fweight = \frac{1}{log(minimaDeviazioneStandard)} * log(FdeviazioneStandard)$$

  con minimaDeviazioneStandard si intende la minima deviazione standard tra tutte le *features considerate*; con FdeviazioneStandard si intende la deviazione standard per la feature F.

- *tollerance*, il valore della tolleranza è inversamente proporzionale alla variabilità delle *feature* rispetto alle altre (per un utente specifico). Più una *feature* è poco variabile più il sistema è tollerante nei suoi confronti in fase di verifica. La tolleranza appartiene all'intervallo [minTol, maxTol]; minTol e maxTol sono due dei parametri che possono essere impostati a priori. La tolleranza vale maxTol se il peso della *feature* corrente è massimo (Fweight=1). Altrimenti, data la lista ordinata dei pesi, la si può calcolare con la seguente formula:

$$Ftollerance = maxTol - i * \frac{maxTol - minTol}{N - 1}$$

  i è la posizione i-esima nella lista ordinata dei pesi. Poiché il peso e la tolleranza sono direttamente proporzionali essi scalano insieme.

In figura 9 segue uno schema riassuntivo delle operazioni eseguite durante la fase di training.

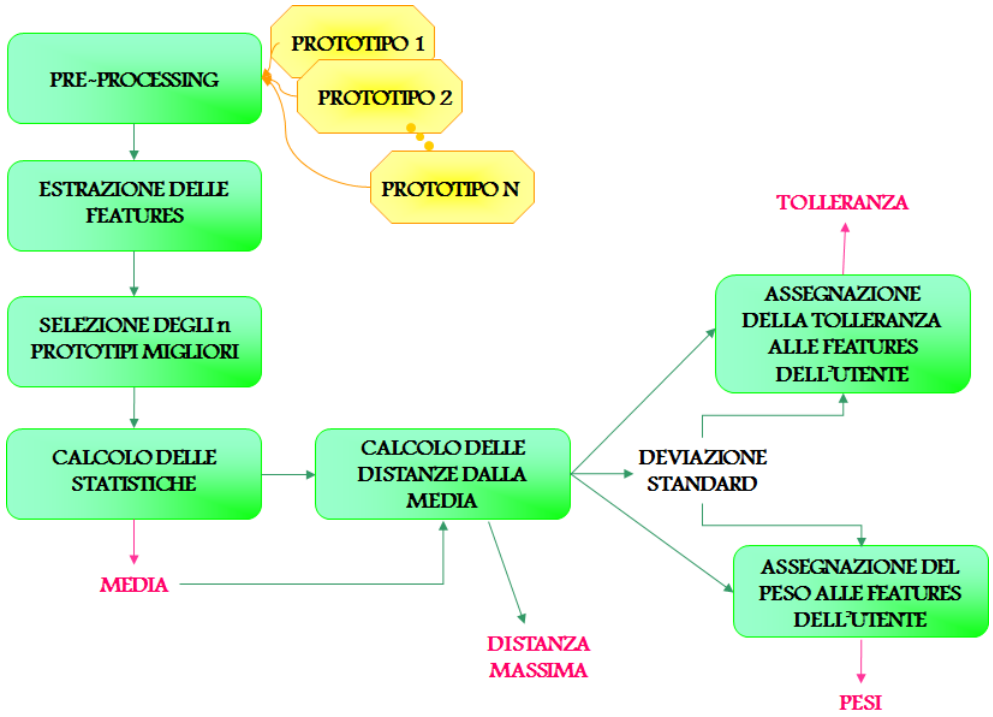



Figura 9: Training per firmatario
Sorgente: Corvallis

### Fase di verifica

Il processo di verifica prevede i seguenti step:

1. Estrazione delle *feature* della firma di cui si vuole verificare l'autenticità, secondo la lista *feature* e *preprocessing* dell'utente che si vuole personificare;
2. Assegnazione dei voti;
3. Calcolo dei punteggi;
4. Calcolo della confidenza.

### Assegnazione dei voti

Il valore di ogni *feature* estratta dalla firma da verificare viene confrontato con la media corrispondente (presa dal *feature centroid vector*). In base a quanto si scosta dalla media, ad ogni *feature* viene assegnato un voto nell'intervallo [-1, 1].

- il voto è uguale a 1 (voto massimo possibile) quando la differenza tra il valore della *feature* e la relativa media salvata è uguale a 0, ossia il valore della *feature* è uguale alla media salvata.
- il voto è uguale a -1 (voto minimo possibile) quando la differenza tra il valore della *feature* e la relativa media salvata è maggiore o uguale a due volte la soglia.

- altrimenti il voto viene assegnato secondo una funzione lineare:

$$Fvoto = \frac{Fthreshold - |FVal - Faverage|}{Fthreshold}$$

### Calcolo dei punteggi

Il core del sistema classifica le firme in base al punteggio che queste riescono a raggiungere. Il punteggio massimo raggiungibile è chiaramente la somma dei pesi delle *feature*. Vediamo ora come si calcola il singolo punteggio che poi andrà a costituire parte del punteggio finale.

$$Fpunteggio = Fvoto * Fweight$$

Fvoto è il voto assegnato alla *feature* F e Fweight è il peso della *feature* F calcolato durante il training.

### Calcolo della confidenza

La decisione finale è data dal valore della confidenza. Se il valore della confidenza è maggiore di 50, la firma viene accettata altrimenti rifiutata. Il calcolo della confidenza è dato da:

$$Confidenza = \frac{0.5 + \sum_{i=1}^{M} FIpunteggio}{2 * \sum_{i=1}^{M} FIweight} * 100$$

M indica il numero di *feature* prese in considerazione.

In figura 10 segue uno schema riassuntivo delle operazioni eseguite durante la fase di verifica.

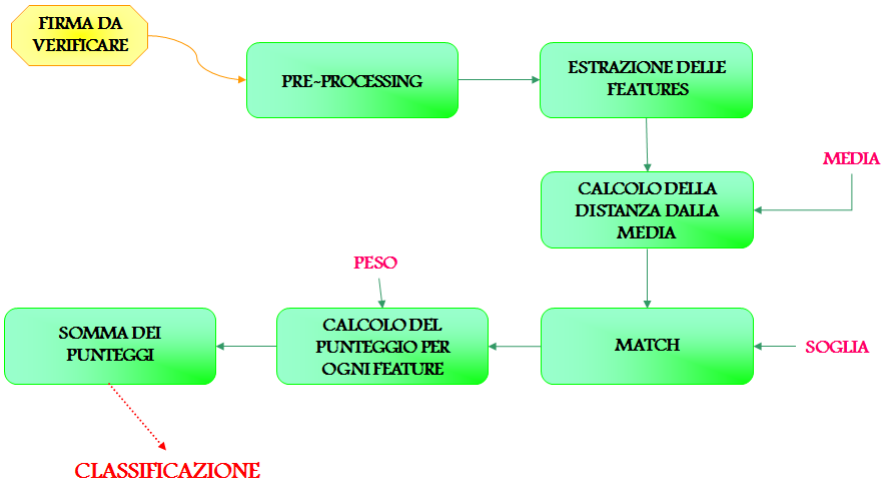



Figura 10: Verifica firma
Sorgente: Corvallis

### Performance

In tabella 1 vengono riportati i risultati di alcuni test.

| FAR random | FAR skilled | FRR |
|---|---|---|
| 0.10 | 0.32 | 0.33 |
| 0.05 | 0.25 | 0.45 |
| 0.00 | 0.09 | 0.45 |
| 0.00 | 0.14 | 0.42 |

Tabella 1: Risultati test prototipo preesistente

**Tecnologie impiegate**

Il prototipo è stato scritto in *Java*, utilizzando l'*Integrated development environment* (*IDE*) *Eclipse*. Le librerie utilizzate per l'elaborazione delle immagini sono *ImageJ* e *ImageMagick*. Il versionamento del codice è stato effettuato grazie a *Content Versioning System* (*CVS*).

**ImageJ**

**ImageMagick**

La prossima sezione introduce il progetto di stage a cui ho preso parte.

## 2.2 Obiettivi

L'obiettivo principale dello stage è lo studio e l'implementazione di un nuovo classificatore di firme statiche, sempre nell'ambito dei classificatori *Statistical-based*. Uno degli obiettivi secondari è testare approfonditamente il software preesistente per identificare le eventuali cause di un bug che rende la classificazione attuale dipendente dalla risoluzione del dispositivo di cattura. L'obiettivo ultimo è quello di valutare le performance del nuovo classificatore e di studiare una modalità di unione delle classificazioni dei due prototipi in modo da aumentare l'accuratezza globale del sistema di verifica delle firme statiche.

## 2.3 Aspettative

L'azienda si aspetta che l'obiettivo principale venga raggiunto.

## 2.4 Vincoli

Non sono stati imposti vincoli particolari per quanto riguarda lo svolgimento del progetto di stage. I vincoli imposti sono:

- utilizzare il linguaggio *Java* per l'implementazione del prototipo
- fornire documentazione adeguata sul prototipo implementato
- utilizzare le librerie *ImageJ* e/o *ImageMagick* per l'elaborazione delle immagini
- lo studio di nuovi metodi di classificazione deve avvenire sempre nell'ambito dei classificatori *Statistical-based*

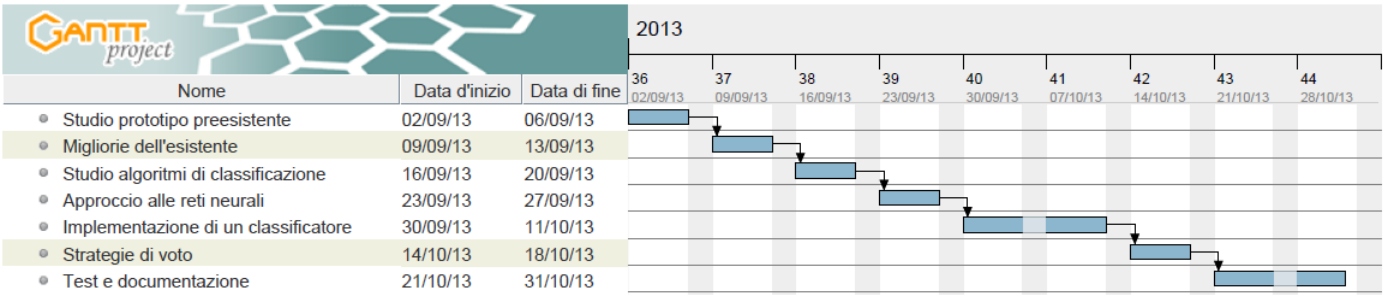
# 3 Attività di stage

Questo capitolo descrive le scelte operative effettuate, i problemi incontrati, gli effetti che questi ultimi hanno avuto sulla pianificazione stilata dal *tutor* aziendale e le soluzioni adottate per raggiungere gli obiettivi del progetto di stage. Per prima cosa introduco la pianificazione originale e il lavoro effettivamente svolto. Successivamente riporto i requisiti che il nuovo prototipo/classificatore dovrà soddisfare. In seguito illustro le decisioni prese in fase di progettazione. In conclusione spiego qualche particolare implementativo.

## 3.1 Pianificazione

Questa parte della relazione si prefigge di mostrare al lettore le discrepanze tra le attività pianificate e le attività effettivamente svolte.

Il piano di lavoro originale, redatto dal *tutor* aziendale, prevedeva le seguenti macro attività suddivise settimanalmente come nel diagramma di *Gantt* riportato in figura 11.



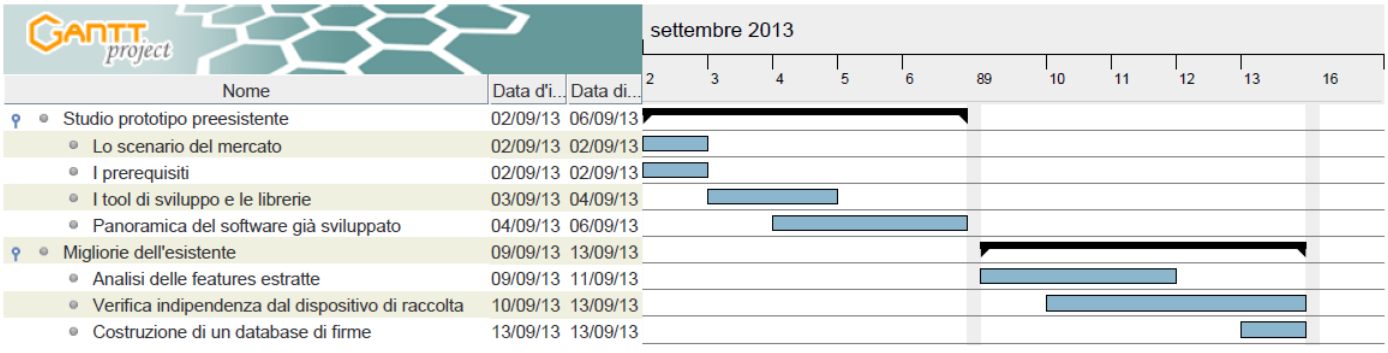

| Nome | Data d'inizio | Data di fine |
|---|---|---|
| Studio prototipo preesistente | 02/09/13 | 06/09/13 |
| Migliorie dell'esistente | 09/09/13 | 13/09/13 |
| Studio algoritmi di classificazione | 16/09/13 | 20/09/13 |
| Approccio alle reti neurali | 23/09/13 | 27/09/13 |
| Implementazione di un classificatore | 30/09/13 | 11/10/13 |
| Strategie di voto | 14/10/13 | 18/10/13 |
| Test e documentazione | 21/10/13 | 31/10/13 |

Figura 11: Diagramma di Gantt, Pianificazione iniziale

Lo stage si è svolto nel periodo che va dal 02/09/2013 al 31/10/2013 per un totale di circa 330 ore. Di seguito entro nel dettaglio di ogni settimana e descrivo il lavoro svolto.

### 3.1.1 Settimane I-II

Le attività previste per le prime due settimane (dal piano di lavoro originale) vengono riportate nel diagramma di *Gantt* in figura 12.




| Nome | Data d'i... | Data di... |
|---|---|---|
| Studio prototipo preesistente | 02/09/13 | 06/09/13 |
| Lo scenario del mercato | 02/09/13 | 02/09/13 |
| I prerequisiti | 02/09/13 | 02/09/13 |
| I tool di sviluppo e le librerie | 03/09/13 | 04/09/13 |
| Panoramica del software già sviluppato | 04/09/13 | 06/09/13 |
| Migliorie dell'esistente | 09/09/13 | 13/09/13 |
| Analisi delle features estratte | 09/09/13 | 11/09/13 |
| Verifica indipendenza dal dispositivo di raccolta | 10/09/13 | 13/09/13 |
| Costruzione di un database di firme | 13/09/13 | 13/09/13 |

Figura 12: Diagramma di Gantt, Settimane I-II

La prima settimana prevedeva attività di studio del dominio applicativo in cui si colloca il progetto di stage.
Vediamo una breve descrizione di ciascuna attività effettuata.

- lo scenario di mercato: il *tutor* aziendale mi ha spiegato che la normativa italiana relativa allo scambio di immagini degli assegni bancari sta per essere aggiornata, in quanto a questi sarà concesso lo scambio di immagini di assegni bancari, e le conseguenze che ne scaturiscono;
- i prerequisiti: le condizioni iniziali sono individuate dalla capacità di un software di estrarre immagini firma da assegni bancari, di classificare le suddette firme in relazione a una base di conoscenza precedentemente acquisita, di essere affidabile;
- i *tool* di sviluppo e le librerie: durante questa attività ho familiarizzato con le tecnologie impiegate nello sviluppo del prototipo sviluppato precedentemente;
- panoramica del software preesistente: nel corso di questa attività ho compreso sommariamente i meccanismi adottati per la classificazione delle immagini firma sia a livello interattivo (utilizzo del software) che a livello di codice implementato.

La seconda settimana prevedeva un'analisi più accurata del software sviluppato precedentemente allo scopo di identificare la causa di un'eventuale bug che rende la classificazione dipendente dalla risoluzione del dispositivo di raccolta immagini (scanner o tavoletta).

Data la mia esperienza nulla nell'elaborazione delle immagini, ho ritenuto opportuno effettuare test utilizzando la tecnica *Walktrough* e non la tecnica *Inspection*. Ho analizzato la maggior parte degli algoritmi di *feature extraction* per individuare un qualche collegamento implicito con la risoluzione dell'immagine processata.
Le mie ricerche non sono riuscite a dimostrare se il software è dipendente o meno dalla risoluzione del dispositivo di acquisizione immagini. Nonostante ciò potrebbero aver individuato leggerissime migliorie da apportare al calcolo del punteggio (paragrafo 2.1.4) di quelle *feature* che assumono valori discreti, poiché per queste, in fase di test, il voto massimo non è raggiungibile e di conseguenza subiscono una penalità.

#### Considerazioni sulle prime due settimane

Le attività previste per le prime due settimane sono state effettuate come pianificato e sono state di fondamentale importanza per l'Analisi dei Requisiti. Tuttavia la seconda settimana non ha portato i risultati sperati.

### 3.1.2 Settimane III-IV

Le attività previste per la terza e la quarta settimana (dal piano di lavoro originale) vengono riportate nel diagramma di *Gantt* in figura 13.

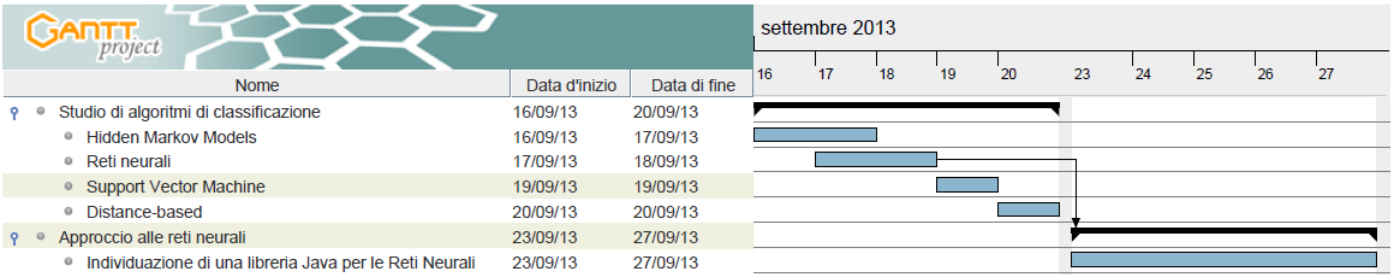



Figura 13: Diagramma di Gantt, Settimane III-IV

La terza settimana prevedeva lo studio critico, attraverso materiale *online* di natura scientifica, dei meccanismi di classificazione di tipo *Statistical-based* più diffusi in modo da stimare in modo opportuno le potenzialità di ciascuno.

Tuttavia l'attività effettivamente svolta è stata lo studio dei *Hidden Markov Model* utilizzati nella classificazione di firme statiche. Poiché, già dai primi articoli analizzati [3][5][6], si evinceva che i *Hidden Markov Models* avrebbero potuto portare dei buoni risultati in termini di *accuracy*, il *tutor* aziendale e io abbiamo deciso di investire le ore (della settimana) rimaste nello studio approfondito di tale modello e di non attenerci al piano di lavoro originale. Inoltre sono stato assente gli ultimi due giorni lavorativi della settimana.

La quarta settimana prevedeva l'approccio alle reti neurali e l'individuazione di una libreria *Java* che ne implementi le meccaniche.

Chiaramente, vista la scelta effettuata durante la terza settimana, questa attività non poteva essere messa in atto. Di conseguenza le operazioni compiute hanno avuto lo scopo di individuare una libreria *Java* che implementa la dinamica dei *Hidden Markov Model*. Dopo aver individuato una libreria ho abbozzato un prototipo usa e getta.

#### Considerazioni sulla terza e quarta settimana

Le attività previste per questo periodo non hanno osservato il piano di lavoro originale. La progettazione iniziale si basa sullo studio effettuato in queste due settimane.

### 3.1.3 Settimane V-VI

Per la quinta e la sesta settimana era prevista l'implementazione di un classificatore in modo prototipale, ma che sarebbe dovuto essere sufficiente a dimostrare la validità del metodo di classificazione scelto.

Le attività compiute durante la quinta e la sesta settimana sono riportate nel diagramma di *Gantt* in figura 14.

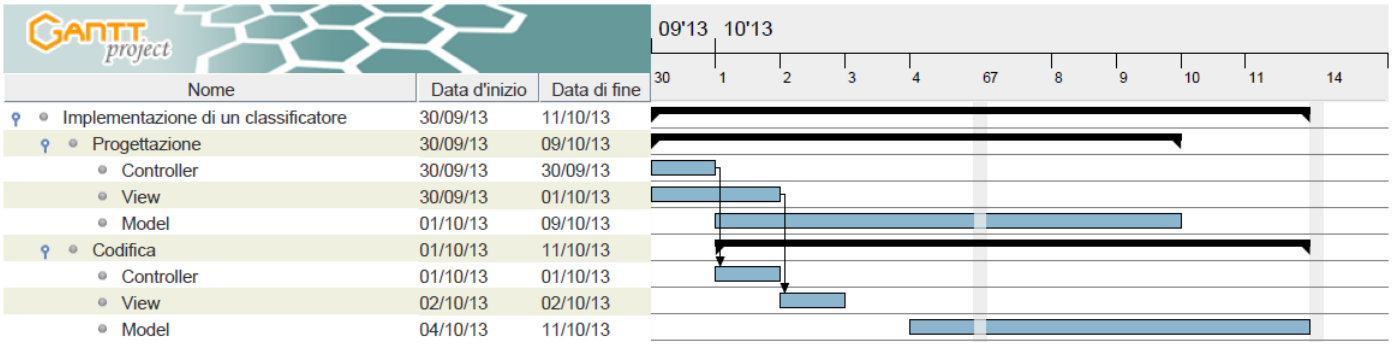



Figura 14: Diagramma di Gantt, Settimane V-VI

Come si può notare nel diagramma di *Gantt* questa fase è stata suddivisa in due sotto-fasi che si sovrappongono:

- Progettazione;
- Codifica;

Le attività di progettazione dei componenti *Controller* e *View* del pattern architetturale *MVC* (paragrafo 3.3.1) hanno richiesto relativamente poco tempo. Questo è dovuto al fatto che è stato possibile riutilizzare parti di architettura del software preesistente. Di conseguenza l'operazione di codifica dei suddetti componenti è stata più un'integrazione che una vera e propria implementazione.

Purtroppo la progettazione del componente *Model*, dato l'ambito esplorativo in cui si colloca il progetto di stage, ha richiesto molto tempo. Questo è dovuto principalmente al ritardo con cui si sono presentati i punti più sensibili relativi all'utilizzo dei *Hidden Markov Models*. Questi sono emersi durante la codifica iniziale e vengono esposti in modo approfondito nella sezione Progettazione (3.3).

**Considerazioni sulla quinta e sesta settimana**

Le attività svolte in questo periodo hanno portato a una più estesa comprensione dei *Hidden Markov Model*. Tuttavia permangono dei problemi con l'algoritmo che permette di addestrare un *Hidden Markov Model* e di conseguenza il prototipo in questo stadio non è stabile.

### 3.1.4 Settimane VII-IX

Le attività previste per l'ultima parte del progetto comprendevano lo studio di una strategia di classificazione unita tra il prototipo realizzato durante lo stage e il software preesistente, i test e la documentazione.

Le attività effettivamente svolte nelle ultime due settimane vengono riportate nel diagramma di *Gantt* in figura 15.

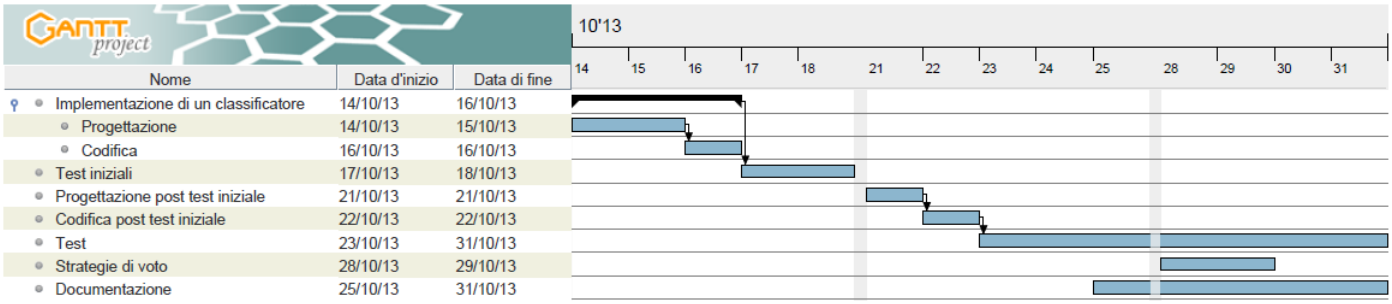



Figura 15: Diagramma di Gantt, Settimane VII-IX

Come si può notare nel diagramma di *Gantt* relativo a questo periodo, nei primi giorni della settima settimana sono riuscito a finire l'implementazione del prototipo. Tuttavia durante i test iniziali ci siamo accorti di un altro problema che avevamo ignorato: alcuni modelli addestrati sono caduti nell'eccessivo adattamento e la loro capacità di categorizzare firme non visionate è stata compromessa.

Durante l'inizio dell'ottava settimana ho studiato come arginare il problema dell'eccessivo adattamento e ne ho implementato una soluzione basata sulla cross-validazione. Test, successive correzioni di bug e documentazione si sono susseguite fino alla fine dello stage.

All'inizio dell'ultima settimana ho studiato e implementato un metodo di classificazione comune che si basa sulle decisioni del prototipo preesistente al progetto di stage e del prototipo sviluppato durante.

#### Considerazioni sul periodo finale

I test sono utili solo se identificano comportamenti non prescritti.

## 3.2 Analisi

Il progetto di stage prevedeva l'implementazione di un nuovo classificatore di firme. In seguito allo studio del dominio applicativo effettuato durante le prime due settimane di stage e alle spiegazioni del *tutor* aziendale sono riuscito ad individuare i requisiti del prototipo richiesto.

Poiché si tratta di un prototipo, esso non sarà collocato nell'ambito applicativo di competenza (applicazione per istituti bancari) e soprattutto non sarà indipendente dal input umano. Dovrà essere un'applicazione locale in grado di addestrare la sua base di conoscenza e grazie all'addestramento dovrà essere in grado di classificare immagini firma. L'analisi dei requisiti è stata effettuata seguendo questi cardini semplicistici.

Di seguito vengono riportati i casi d'uso principali individuati.

### 3.2.1 Casi d'uso

I casi d'uso (*use case*) sono tecniche per individuare i requisiti funzionali. Essi descrivono interazioni del sistema e degli attori esterni al sistema e come il sistema deve essere utilizzato [7]. L'attore principale è uno solo: utente che configura e avvia il prototipo a piacimento.

#### UC1 Caso d'uso generale

**Scopo e descrizione**: un utente deve poter selezionare un database di firmatari e rispettive firme. Una volta selezionato il database, deve essere possibile effettuare il training impostando il numero di firme da utilizzare e i preprocessing. Ultimato il training deve essere possibile testare l'accuratezza del sistema classificando le firme restanti nel database.

**Precondizione**: il sistema si trova nello stato iniziale.

**Postcondizione**: il sistema restituisce e salva su file i risultati relativi alla classificazione effettuata.

**Scenario principale**:

- l'utente seleziona il database di firme su cui lavorare;
- l'utente configura i parametri disponibili;
- l'utente avvia il training;
- l'utente avvia il testing.

#### UC1.3 Avvia training

**Scopo e descrizione**: un utente deve poter avviare il training su un database con impostazioni a piacimento.

**Precondizione**: il sistema conosce il database selezionato dall'utente e i parametri che quest'ultimo ha impostato precedentemente.

**Postcondizione**: il sistema addestra la sua base di conoscenza secondo la configurazione dell'utente e la salva su file.

**Scenario principale**:

- l'utente avvia il training.

#### UC1.4 Avvia testing

**Scopo e descrizione**: un utente deve poter avviare il testing sulle firme (di un database) non visionate in fase di training.

**Precondizione**: il sistema conosce il database selezionato dall'utente e la base di conoscenza del database è stata addestrata in precedenza.

**Postcondizione**: il sistema restituisce i risultati della classificazione e li salva su file.

**Scenario principale**:

- l'utente avvia il testing.

Il diagramma *use case* che descrive le macro funzionalità del prototipo segue in figura 16.

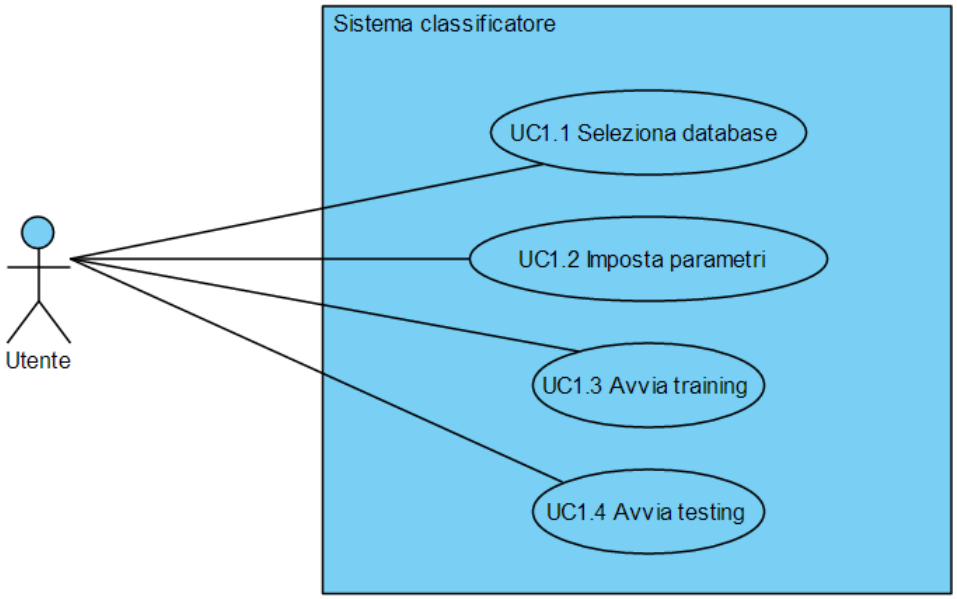



Figura 16: Diagramma use case, UC1 caso d'uso generale

### 3.2.2 Requisiti

La classificazione dei requisiti sarà conforme al seguente formalismo:

$$R\{TIPO\}\{IMPEGNO\}\{GERARCHIA\}$$

dove:

- **TIPO** può essere:
  - *F*, indica un requisito funzionale
  - *Q*, indica un requisito di qualità
  - *V*, indica un requisito di vincolo
- **IMPEGNO** può essere:
  - *O*, indica un requisito obbligatorio, ossia il soddisfacimento di questo vincolo è necessario per la realizzazione del prodotto finale
  - *D*, indica un requisito desiderabile, ossia un requisito auspicabile per un prodotto di qualità
- **GERARCHIA**: i requisiti sono organizzati gerarchicamente secondo una struttura ad albero. Un requisito (padre) può essere complesso al punto di dover essere suddiviso in requisiti minori (figli). X.Y individua un requisito Y figlio di un requisito X.

I requisiti determinati a partire dall'obiettivo principale del progetto di stage vengono riportati in tabella 2.

Tabella 2: Requisiti

| **Requisito** | **Descrizione** | **Caso d'uso** |
|---|---|---|
| RFO1 | Il sistema deve essere in grado di classificare in modo corretto immagini di firma autografa in seguito a una fase di training eseguita conformemente alle impostazioni utente, su un database selezionato precedentemente. | UC1 |
| RFO1.1 | Il sistema deve permettere all'utente di selezionare un database di firmatari con le rispettive firme. | UC1.1 |
| RFO1.2 | Il sistema deve poter caricare in memoria il file di configurazione dei parametri impostato dall'utente. | UC1.2 |
| RFO1.3 | Il sistema deve permettere all'utente di effettuare il training su un database selezionato precedentemente. | UC1.3 |
| RFO1.3.1 | Il sistema deve essere in grado di elaborare il template/firma media di un firmatario a partire da N firme genuine del sottoscrittore. | UC1.3 |
| RFO1.3.2 | Il sistema deve essere in grado di salvare il template su file. | UC1.3 |
| RFO1.4 | Il sistema deve essere in grado di classificare correttamente firme che non ha mai visionato. | UC1.4 |
| RFO1.4.1 | Il sistema deve essere in grado di caricare in memoria il template/firma media del sottoscrittore la cui firma si sta testando. | UC1.4 |
| RFO1.4.2 | Il sistema deve essere in grado di confrontare la firma sotto test con il template del sottoscrittore. | UC1.4 |
| RFO1.4.3 | Il sistema deve essere in grado di prendere una decisione sul risultato del confronto e quindi accettare o rifiutare la firma sotto test. | UC1.4 |
| RFO1.4.4 | Il sistema deve essere in grado di riportare a video e salvare su file il risultato della classificazione. | UC1.4 |
| RFV2 | Il core del sistema deve essere un classificatore di tipo *Statistical-based*. | |
| RFQ3 | Il sistema deve garantire un'*accuracy* media del 80%. | |

*Tabella 2: si conclude dalla pagina precedente*

Il test sul output dell'attività di analisi ha compreso test di natura statica sui requisiti. In particolare ho verificato la loro atomicità, chiarezza di esposizione e completezza.

## 3.3 Progettazione

Questa parte della relazione vuole mostrare al lettore le scelte progettuali effettuate relative all'architettura e alle tecnologie da impiegare per soddisfare i requisiti individuati in fase di analisi.

### 3.3.1 Architettura

L'architettura è l'output della fase di progettazione, essa ha l'obiettivo di rendere l'attività di programmazione il più vincolata possibile senza lasciare spazio all'interpretazione.

Poiché il software preesistente utilizza il pattern architetturale *Model View Controller* ho deciso di adoperarlo anche io. I vantaggi principali di un'architettura *MVC* si possono riassumere nei seguenti punti:

- consente di tenere una netta separazione tra le componenti di stato (*Model*), i componenti mostrati sullo schermo (*View*) e il loro comportamento in relazione agli eventi (*Controller*)
- manutenibilità del software
- modularità
- leggibilità

In figura 17 è possibile visionare il pattern MVC.

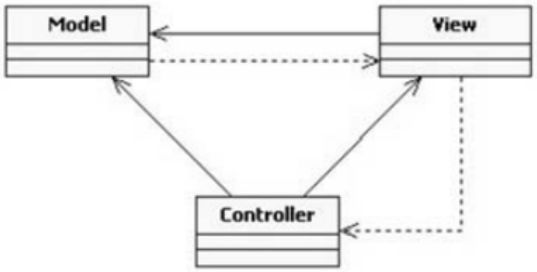



Figura 17: Pattern MVC

### 3.3.2 Hidden Markov Models

Questa sezione effettua una panoramica sui *Hidden Markov Models*, spiegandone le dinamiche, la modellazione delle firme, i problemi incontrati e le soluzioni adottate per superarle.

Segue un'introduzione ai modelli di Markov.

#### Processo di Markov

Un processo (sistema) di Markov è contraddistinto dalle seguenti affermazioni:

- ha N stati distinti : $S_1$, $S_2$, ..., $S_N$;
- è caratterizzato da passi discreti, t=1, t=2, ...
- la probabilità di partire da un determinato stato è data dalla matrice stocastica:

$$\Pi = \{\pi_i\} : \pi_i = P(q_1 = s_i) \; con \; 1 \leq i \leq N, \; \pi_i \geq 0 \; e \; \sum_{i=1}^{N} \pi_i = 1$$

- al t-esimo istante il processo si trova esattamente in uno degli stati a disposizione, indicato dalla variabile $q_t$:
$$q_t \in \{S_1, S_2, ..., S_N\}$$
- ad ogni iterazione, lo stato successivo viene scelto con una determinata probabilità. Tale probabilità è solo ed esclusivamente dipendente dallo stato precedente, e non dalla sequenza di stati che lo ha preceduto (*memoryless*). Questa dichiarazione riassume la *proprietà di Markov*:
$$P(q_{t+1} = s_j \mid q_t = s_i,\ q_{t-1} = s_k,\ ...,\ q_1 = s_1) \ = \ P(q_{t+1} = s_j \mid q_t = s_i)$$
- le probabilità di transizione tra stati (invarianti nel tempo) sono descritte dalla matrice stocastica:
$$A = \{a_{ij}\} \ : \ a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$$

**Modello di Markov**

Date le caratteristiche del processo di Markov descritte nella sezione precedente un modello di Markov si può riassumere con:
$$\lambda = (A,\ \pi)$$
Le operazioni interessanti che si possono effettuare sui modelli di Markov sono:

- addestramento o training (si costruiscono gli elementi costituenti del modello);
- interrogazioni sul modello per esempio calcolare la probabilità di una sequenza di stati, dato il modello.

I modelli di Markov modellano processi Markoviani in cui gli stati sono espliciti e osservabili: c'è una corrispondenza biunivoca tra le osservazioni e i stati emittenti. Purtroppo nella realtà esistono modelli i cui stati non sono espliciti, essi sono identificabili unicamente tramite le osservazioni, in maniera probabilistica. All'osservatore è accessibile soltanto una sequenza di simboli in base alla quale egli può inferire soltanto la probabilità degli stati corrispondenti. Questi modelli vengono denominati modelli di Markov a stati nascosti.

**Hidden Markov Model**

Il modello di Markov a stati nascosti può essere inteso come un' estensione del modello di Markov che si distingue per la non osservabilità dei suoi stati. Esso si compone di:

- un insieme $S=\{S_1,S_2,...,S_N\}$ di stati nascosti;
- una matrice di transizione tra stati nascosti:
$$A = \{a_{ij}\} \ : \ a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$$
- una matrice di probabilità di partire da uno stato iniziale nascosto:
$$\Pi = \{\pi_i\} : \pi_i = P(q_1 = s_i) \ con \ 1 \leq i \leq N,\ \pi_i \geq 0 \ e \ \sum_{i=1}^{N} \pi_i = 1$$
- un insieme $V=\{V_1,V_2,...,V_M\}$ di simboli di osservazione;

- al t-esimo istante il processo emette uno fra i simboli a disposizione, indicato dalla variabile $o_t$:
$$o_t \in \{V_1, V_2, ..., V_M\}$$
- una matrice stocastica con le probabilità di emissione, indica la probabilità di osservare il simbolo $V_k$ quando il sistema si trova nello stato $S_j$:
$$B = \{b_j(k)\} \ : b_j(k) = P(o_t = k \mid q_t = j)$$

In generale un *HMM* si indica con la tripla
$$\lambda = (A,\ B,\ \pi)$$

**Tipi di *Hidden Markov Models***

I Hidden Markov Models possono venire classificati in base alla topologia degli stati o in base a cosa rappresentano i simboli.

Topologie:

- modelli ergodici, se è possibile andare da ogni stato i ad ogni stato j, non necessariamente in un solo step;
- modelli *Left to Right* o *Bakis*, se la matrice delle probabilità di transizione è una matrice triangolare superiore (l'indice degli stati non decresce col tempo). Questa topologia viene adottata per la modellazione delle firme perché si assume che la scrittura latina vada da sinistra verso destra.

In figura 18 è possibile visualizzare le due topologie.

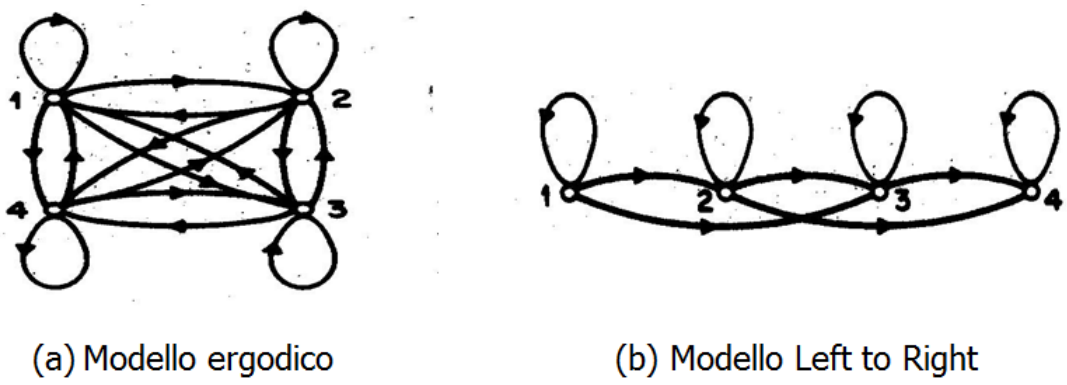



Figura 18: Topologie HMM

Nel *discrete HMM* i simboli sono discreti, mentre nel *continuous HMM* i simboli sono vettori. Nel primo caso, $B_j$ è una matrice, mentre nel secondo caso $B_j$ è una *mixture of gaussian probalility density functions.*

**I tre problemi dei *Hidden Markov Models*[8]**

1. ***Evaluation***, dato un modello di Markov a stati nascosti $\lambda$ e una sequenza d'osservazione O, trovare la probabilità che la sequenza O possa venire generata dal modello $\lambda$ : P(O|$\lambda$);
2. ***Decoding***, dato un HMM $\lambda$ e una sequenza O, trovare la sequenza di stati $\pi_{\mathrm{i}}$ che massimizza P(O, $\pi_{\mathrm{i}}$|$\lambda$);
3. ***Learning***, dato un modello $\lambda$ con transizioni ed emissioni non note e una sequenza d'osservazione O trovare le matrici A e B tali che P(O|$\lambda$) è massimo. In altre parole vorremmo usare la sequenza d'osservazione per addestrare un *HMM*.

**Soluzioni ai problemi individuati[8]**

Poiché in generale i 3 problemi sono computazionalmente intrattabili, delle soluzioni basate sulla programmazione dinamica sono state adottate.

1. ***Forward algorithm***, poiché sommare le probabilità di tutti i possibili modi in cui si può generare la sequenza O coinvolge un numero esponenziale di cammini $\pi$ si definisce probabilità *forward*:
$$f_{\mathrm{t}}(i) = P(O_1 ... O_{\mathrm{t}},\ \pi_{\mathrm{t}} = i)$$
può essere derivata ricorsivamente:
   - caso base
   $$f_1(i) = P(\pi_{1=\mathrm{i}} * b_{\mathrm{i}}(O_1)\ con\ 1 \leq\ i \leq\ k$$
   - passo induttivo
   $$f_{\mathrm{t}+1}(j) = [\ \sum_{i=1}^{k} f_{\mathrm{t}}(i) * a(i,j)\ ] * b_{\mathrm{j}}(O_{\mathrm{t}+1})$$
2. ***Viterbi's algorithm***, per una descrizione si rimanda al documento[8];
3. ***Baum-Welch algorithm***, algoritmo di *expectation-maximization* basato sul *forward-backward algorithm* e *Viterbi's algorithm*. Può convergere a massimi locali, dipende dai valori assunti dalle matrici iniziali.

I campi in cui vengono applicati i *HMM* sono molteplici. Di seguito riportiamo alcuni di questi:

- riconoscimento della parola;
- sintesi vocale;
- riconoscimento texture;
- riconoscimento movimento del corpo;
- riconoscimento della grafia.

**Modello HMM che descrive le firme di un sottoscrittore**

Vediamo ora come ho inizializzato e addestrato un modello di Markov a stati nascosti che descrive le firme di un sottoscrittore.

La scelta del numero di stati nascosti che si assume siano presenti è un punto fondamentale. In letteratura hanno utilizzato due diversi approcci:

- in [3] suggeriscono di non impostare il numero di stati nascosti a priori ma di calcolarlo in base alla larghezza della firma;
- in [5] decidono a priori il numero di stati nascosti e lo impostano a 4.

Per semplicità ho scelto la seconda strada.

Ricordiamo che un *HMM* è caratterizzato dalla tripla $\lambda$=(A,B,$\pi$). Come abbiamo accennato in precedenza la topologia che meglio descrive le firme latine (scritte da sinistra verso destra) è la *Left to Right*. L'uso di questa topologia rende l'inizializzazione delle matrici A e $\pi$ banale. Come indicato nel documento [3] ho aggiunto un ulteriore vincolo sulla matrice delle probabilità di transizione (A): i salti di stato non sono ammessi.

Vediamo le matrici A e $\pi$ risultanti:

$$A = \begin{vmatrix} 0.95 & 0.05 & 0 & 0 \\ 0 & 0.95 & 0.05 & 0 \\ 0 & 0 & 0.95 & 0.05 \\ 0 & 0 & 0 & 1.0 \end{vmatrix}$$

$$\pi = \begin{vmatrix} 1.0 & 0 & 0 & 0 \end{vmatrix}$$

Per quanto riguarda la matrice delle probabilità di emissione ho scelto di utilizzare i *discrete HMM*.

Vediamo ora i passi progettati per l'inizializzazione della matrice B:

- *feature extraction*, poiché servono delle *features* ho deciso di testare la *feature axial slant* (come in [3]) e la *feature Discrete Cosine Transform (DCT)* (come in [5]).
- tipi di *preprocessings*, ho deciso di testare la segmentazione a griglia (come in [3]) per la *feature axial slant* e la segmentazione basata sul centro di gravità per la *feature DCT*[5]. Entrambe le segmentazioni dividono l'immagine firma in 64 celle.
- la fase di *feature extraction* restituisce 64 valori. Questi costituiscono 4 vettori d'osservazione per firma.
- i punti sopra descritti vanno effettuati per tutte le firme che si intendono utilizzare nella fase di training
- i vettori ottenuti vengono dati in pasto al metodo KMeansLearner della libreria JScience, il quale effettua il *clustering* e inizializza la matrice B.

In figura 19 è disponibile una figura che riporta i passi eseguiti per individuare i vettori d'osservazione.

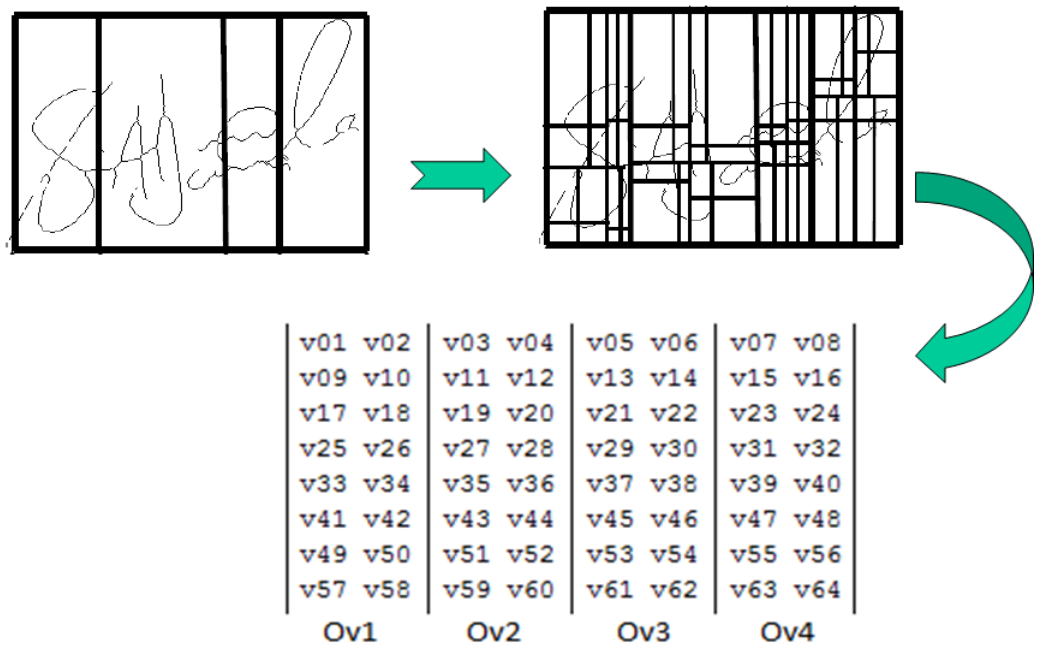



Figura 19: Dalla firma ai vettori d'osservazione

Per quanto riguarda l'addestramento del modello, basta applicare l'algoritmo di Baum-Welch.

### 3.3.3 Tecnologie utilizzate

Di seguito vengono elencate le tecnologie utilizzate per lo sviluppo del prototipo.

#### *Java*

Ho utilizzato il linguaggio *Java* come richiesto dall'azienda. Fra i vantaggi che il linguaggio *Java* offre ricordiamo:

- compatibilità multipiattaforma (grazie alla *Virtual Machine*);
- velocità di sviluppo;
- grande disponibilità di librerie;
- alta integrazione con il web;
- incoraggia il riutilizzo del codice.

#### *Eclipse*

Ho utilizzato l'ambiente di sviluppo *Eclipse* come richiesto dall'azienda. I vantaggi di questo vincolo si possono riassumere in:

- navigazione intelligente del codice;
- *code completion* integrata;
- compilazione al volo.

#### *ImageJ*

Per l'elaborazione delle immagini ho utilizzato la libreria *ImageJ* perché è di facile comprensione e non richiede uno step in più per il *preprocessing* delle immagini, quale l'utilizzo del terminale.

#### Librerie *Java* che implementano i *HMM*

In seguito alla decisione di adoperare i *Hidden Markov Models* per la classificazione di firme statiche, abbiamo effettuato una *software selection*. Di seguito elenco le librerie *Java* che vengono utilizzate per i *HMM* e descrivo i problemi incontrati.

1. **Jahmm**
   - implementa gli algoritmi relativi ai *Hidden Markov Models*, quali *forward algorithm*, *Viterbi's algorithm*, *Baum-Welch algorithm*
   - è capace di trattare sia i *discrete HMM* che i *continuous*;
   - possiede anche versioni *scaled* degli algoritmi *forward* e *Baum-Welch* allo scopo di evitare l'*underflow*;
   - poiché è ben documentata è stata la prima libreria scelta per trattare i *HMM*;
   - problema incontrato: in fase di codifica mi sono accorto di problemi di *underflow* durante il calcolo della matrice delle probabilità di emissione iniziale.
   - conseguenza: ho dovuto cercare un'altra libreria.
2. **JScience**
   - anche questa libreria implementa gli algoritmi relativi ai *HMM*;
   - è stata la seconda libreria scelta per trattare i *HMM*;
   - problema incontrato: l'algoritmo di Baum-Welch non è in grado di massimizzare modelli utilizzando molteplici sequenze d'osservazione, caratteristiche della fase di addestramento di un modello *Left to Right*;
   - soluzioni cercate: implementare un algoritmo di Baum-Welch per i *HMM* di tipo *Left to Right* seguendo il documento[**?**, **?**, 9] portato alla luce dal professor Finesso. Tuttavia la soluzione implementata o era dominata da errori di codifica o soffriva anch'essa di *underflow*.
   - conseguenza: ho dovuto cercare un'altra libreria per quanto riguarda l'algoritmo di Baum-Welch.
   - la libreria viene utilizzata per il metodo KMeansLearner, il quale dati in input N vettori, e il numero di stati nascosti che si assume il modello abbia, effettua il *clustering* sugli N vettori e inizializza la matrice B del modello.
3. **jhmm**
   - anch'essa implementa gli algoritmi relativi ai *HMM*
   - pregi: il suo algoritmo di Baum-Welch è in grado di utilizzare molteplici sequenze d'osservazione per massimizzare il modello voluto, tutti i calcoli che svolge sono in *log-space* (somme di logaritmi sono più accurate di moltiplicazioni di numeri piccoli);
   - limitazioni: è una libreria poco documentata, tratta solo *Hidden Markov Models* di tipo *discrete*.

## 3.4 Implementazione

Il codice è stato scritto seguendo per quanto possibile la progettazione e documentando adeguatamente. In fase di programmazione ho eseguito test non programmati che hanno portato al ritrovamento di diversi bug presenti sia in librerie esterne sia in codice da me scritto. I bug e le mancanze delle librerie hanno bloccato l'attività di programmazione e mi hanno costretto a tornare alla fase di progettazione.

# 4 Valutazione retrospettiva

## 4.1 Copertura dei requisiti

### 4.1.1 Possibili sviluppi alle attività svolte

## 4.2 Conoscenze acquisite

## 4.3 Distanza tra conoscenze richieste e conoscenze possedute

# Glossario

**dematerializzazione** processo che ha come obiettivo ultimo la creazione di un flusso di documenti digitali aventi pieno valore giuridico, che vada prima ad affiancare e poi, sul lungo periodo, a sostituire la normale documentazione cartacea presente negli archivi di qualunque attività pubblica o privata.. 11

**matrice stocastica** matrice le cui righe sommano a 1.. 31

**SOA** un'architettura software adatta a supportare l'uso di servizi Web per garantire l'interoperabilità tra diversi sistemi così da consentire l'utilizzo delle singole applicazioni come componenti del processo di business e soddisfare le richieste degli utenti in modo integrato e trasparente.. 8

# Riferimenti bibliografici

[1] Hemanta Saikia, Kanak Chandra Sarma *Approaches and issues in Offline Signature Verification System.*

[2] Battista L., Rivard D., Sabourin R., Granger E., Maupin P., *State of the art in off-line signature verification.*

[3] Justino, Yacoubi, Bortolozzi, Sabourin *An Off-Line Signature Verification System Using HMM and Graphometric Features.*

[4] Yazan M. Al-Omari, Khairuddin Omar *State-of-the-Art in Offline Signature Verification System.*

[5] Adebayo Daramola, Samuel Ibiyemi *Offline Signature Recognition using Hidden Markov Model (HMM).*

[6] Behrouz Vaseghi, Somayeh Hashemi *Offline signatures Recognition System using Descrete Cosine Transform and VQ/HMM.*

[7] *Corso di Ingegneria del Software, facoltà di Informatica, Università di Padova.*

[8] L.R. Rabiner, B.H. Juang *An Introduction to Hidden Markov Models.*

[9] Levinson, Rabiner, Sondhi *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition.*