

# POLITECNICO DI TORINO

**Corso di Laurea  
in Matematica per l'Ingegneria**

Tesina di Matematica per l'Intelligenza Artificiale

Davide Furfaro 309316  
Ulderico Guzzardi 307530

Anno Accademico 2024/25



# Indice

<b>1 Analisi del dataset</b>	5
1.1 Introduzione al dataset . . . . .	5
1.2 Descrizione delle features . . . . .	6
1.3 Suddivisione del dataset . . . . .	7
1.4 Distribuzione dei dati . . . . .	7
1.5 Relazioni tra i dati . . . . .	8
<b>2 PCA</b>	11
2.1 Introduzione . . . . .	11
2.2 Il metodo . . . . .	12
2.2.1 Dettagli matematici . . . . .	12
2.2.2 Ricerca delle componenti principali . . . . .	13
2.3 Applicazione al dataset . . . . .	15
<b>3 FDA</b>	21
3.1 Introduzione . . . . .	21
3.2 Il metodo . . . . .	21
3.3 Metodi di classificazione . . . . .	24
3.3.1 LDA . . . . .	24
3.3.2 QDA . . . . .	25
3.4 Applicazione al dataset . . . . .	25
<b>4 SVM</b>	27
4.1 SVM lineari . . . . .	27
4.1.1 Dettagli matematici . . . . .	27
4.1.2 Applicazione al dataset . . . . .	31
4.2 SVM non lineari . . . . .	36

4.2.1	Dettagli matematici	36
4.2.2	Applicazione al dataset	38
<b>5</b>	<b>MLP</b>	<b>41</b>
5.1	Dettagli matematici	41
5.2	Applicazione al dataset	43
<b>6</b>	<b>Conclusioni</b>	<b>47</b>



# Capitolo 1

## Analisi del dataset

### 1.1 Introduzione al dataset

Il dataset scelto proviene dal database di una compagnia di telecomunicazioni iraniana e rappresenta un campione casuale di 3150 clienti diversi, i cui dati sono stati raccolti nel corso di 12 mesi.

In particolar modo, ogni cliente è descritto da 13 features di tipo intero (non ci sono valori mancanti), e classificato da una variabile binaria chiamata *churn*, che ci dice se il cliente ha abbandonato il piano tariffario dopo 12 mesi.

I 12 mesi di raccolta dei dati sono stati suddivisi nel seguente modo:

- Primi 9 mesi per la raccolta dei dati di ogni cliente relativamente alle 13 features;
- Ultimi 3 mesi designati come "planning gap", ovvero come periodo sfruttato dal cliente per fare le sue valutazioni e decidere se abbandonare o meno;
- 12° mese, esito della scelta del cliente.

Questo tipo di dataset si presta a studi relativi alla correlazione (se c'è) tra le varie features, i loro valori e la scelta del cliente di abbandonare o meno. Sulla base di questi dati, per esempio, si potrebbe costruire un modello di classificazione binaria in modo da individuare quali clienti sono a rischio di abbandono, intervenendo per tempo proprio sulle problematiche più urgenti relative a un determinato cliente.

## 1.2 Descrizione delle features

Come già scritto, ogni cliente è descritto da 13 features di tipo intero, vediamole in modo più approfondito:

- **Customer ID:** identificatore anonimo del cliente, numero intero che va da 0 a 3149. Chiaramente non è una vera e propria feature, tanto da non essere conteggiata tra le 13;
- **Call Failures:** numero di chiamate fallite dal cliente;
- **Complains:** variabile binaria pari a 0 se il cliente non ha effettuato reclami, e pari a 1 altrimenti;
- **Subscription Length:** durata totale del contratto, misurata in numero di mesi. Notare che la durata dei contratti, nonostante i dati siano stati raccolti nel corso di 9 mesi, può anche superare i 3 anni in certi casi;
- **Charge Amount:** descrive la spesa del cliente in una scala da 0 (valore minimo) a 9 (valore massimo);
- **Seconds of Use:** Somma totale di secondi di utilizzo per le chiamate;
- **Frequency of Use:** numero totale di chiamate effettuate;
- **Frequency of SMS:** numero totale di SMS inviati;
- **Distinct Called Numbers:** totale di numeri di telefono distinti chiamati
- **Age Group:** attribuisce il cliente ad una fascia di età. Il valore di questa feature è compreso tra 1 (cliente molto giovane) e 5 (cliente molto anziano);
- **Tariff Plan:** variabile binaria che individua il piano tariffario del cliente. Se il cliente paga in anticipo (pay as you go = prepagato) per minuti e SMS, allora la variabile è pari a 1. Se, invece, il cliente sottoscrive un contratto (per esempio mensile), allora questa feature avrà valore 2;

- **Status**: altra feature binaria, pari a 1 se lo stato dell'utente è attivo, e uguale a 2 altrimenti;
- **Age**: età del cliente;
- **Customer Value**: valore del cliente per l'azienda
- **Churn**: è ciò che classifica il cliente, è uguale a 0 se non ha abbandonato, 1 altrimenti.

### 1.3 Suddivisione del dataset

Abbiamo deciso di suddividere il dataset come segue:

- **Training set = 65%**: frazione di dati utilizzata per allenare i modelli;
- **Test set = 35%**: frazione di dati utilizzata per valutare le prestazioni finali del modello.

Tutti i set sono stati suddivisi in maniera stratificata, mantenendo la stessa distribuzione delle classi del dataset da cui sono stati costruiti.

### 1.4 Distribuzione dei dati

Analizzando come i dati si distribuiscono nelle due differenti classi, si nota immediatamente un evidente sbilanciamento verso la classe Churn = 0, cioè quella dei clienti che alla fine dei 12 mesi hanno deciso di non abbandonare. Come vediamo nella Figura 1.2 in basso, circa l'84% dei clienti appartiene alla classe 0 (2655 su un totale di 3150), mentre il restante 16% appartiene alla classe 1. Questa discrepanza si riflette chiaramente anche nel test set, in quanto le proporzioni tra le due classi nei vari set sono rimaste invariate. Per evitare che, durante l'apprendimento, questo sbilanciamento danneggi in maniera eccessiva l'accuratezza dei metodi utilizzati successivamente per la classificazione, abbiamo effettuato due ricampionamenti diversi <sup>1</sup> dei dati. Il primo, tramite il

---

<sup>1</sup>Entrambi i metodi fanno parte della libreria *imblearn* di Python, progettata apposta per essere integrata coi metodi di scikit-learn

metodo *RandomOverSampler*, non fa altro che clonare casualmente i dati della classe in minoranza in modo da bilanciare i campioni. Il secondo ricampionamento, invece, è stato fatto tramite il metodo *SMOTE* (*Synthetic Minorit Over-sampling TEchnique*), che genera nuovi dati appartenenti alla classe minoritaria partendo da quelli del database originale. Per generare i nuovi dati, vengono effettuate delle interpolazioni tra elementi "vicini" appartenenti alla stessa classe, fino a quando non si arriva al totale bilanciamento dei campioni. Teoricamente, ci si aspetta che il secondo metodo sia più efficiente, ma le osservazioni al riguardo verranno trattate nei capitoli successivi. Bisogna chiaramente sottolineare che, avendo aumentato soltanto le dimensioni del training set (per evitare data leak), nei due dataset col ricampionamento vi sono proporzioni diverse tra training set e test set. Infatti, circa il 76% dei nuovi dati fa adesso parte del training set, mentre il restante 24% costituisce il test set.

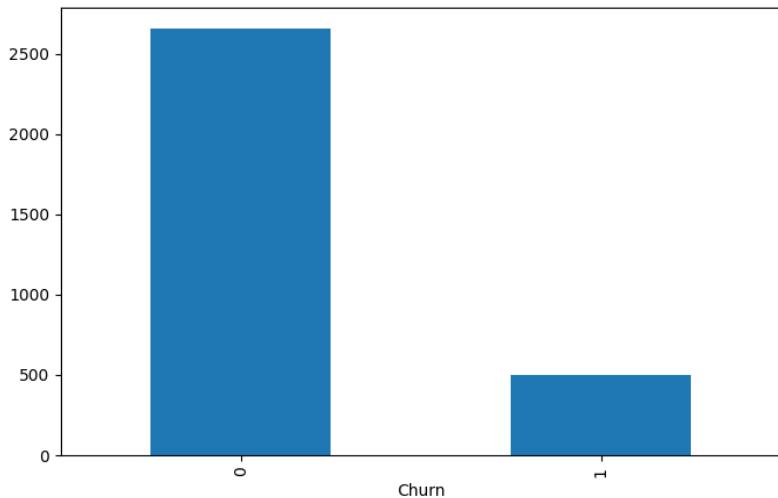


Figura 1.1: Distribuzione iniziale del dataset nelle due classi

## 1.5 Relazioni tra i dati

Dando uno sguardo immediato alle features, vediamo come alcune di esse abbiano una correlazione più che evidente. Per esempio, la variabile "Age Group" è necessariamente correlata alla variabile "Age", stesso discorso valido per "Frequency of Use" e "Seconds of Use", anche se in questo caso la correlazione potrebbe essere meno diretta.

In ogni caso, per studiare come le varie features interagiscono tra loro, è

stata calcolata la matrice di correlazione, che risulterà molto utile anche per la proiezione dei dati tramite PCA (argomento trattato nel prossimo capitolo).

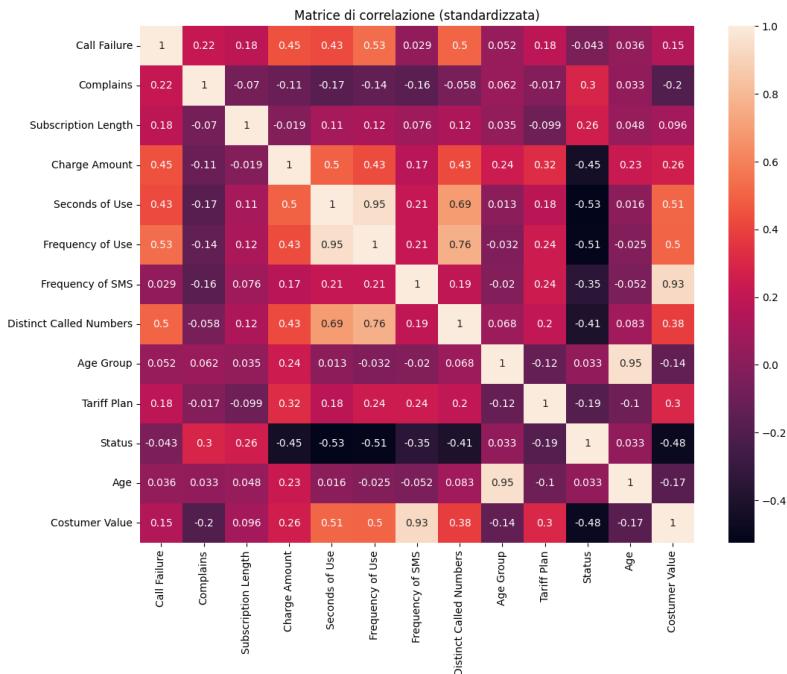


Figura 1.2: Matrice di correlazione delle variabili

Oltre a ciò che ci aspettavamo, ci sono altre interazioni piuttosto interessanti tra le variabili. Innanzitutto, è comprensibile che le features "Frequency of Use" e "Seconds of Use" siano positivamente correlate con la variabile "Distinct Called Numbers". È altrettanto comprensibile che la variabile binaria "Status" sia correlata negativamente con tutte le variabili che riguardano le chiamate, i secondi di utilizzo ecc. e, ovviamente, con il "Customer Value". Ricordiamo infatti che se il cliente è attivo allora lo status è pari a 1, viceversa la variabile assume valore 2. Infine, un'ultima relazione molto importante da notare è quella tra le features "Frequency of SMS" e il "Customer Value" (coefficiente di correlazione pari a +0.93), il che ci suggerisce che l'azienda iraniana trova maggiori entrate proprio grazie ai messaggi.

# Capitolo 2

## PCA

### 2.1 Introduzione

Quando si ha a che fare con dei dataset che presentano un elevato numero di features, può essere estremamente utile ridurre la dimensionalità per rendere più efficaci i modelli di apprendimento. Nel nostro caso, i dati sono situati in uno spazio di dimensione 13, quindi ridurre il numero di variabili serve anche a visualizzare meglio come questi sono distribuiti, cercando di vedere anche quanto le due classi sono separate. Uno dei metodi che realizzano questa riduzione è la **Principal Components Analysis** (PCA). La PCA è un metodo di *unsupervised learning*, poiché cerca di trovare una nuova rappresentazione dei dati senza sapere nulla sulle classi a cui essi appartengono. Lo scopo di questo metodo è quello di ridurre la dimensionalità minimizzando allo stesso tempo la perdita di informazioni; in modo tale che, nel caso in cui si volesse tornare al dataset originale, lo si possa fare senza troppi danni. Quello che fa la PCA è costruire un nuovo set di variabili "artificiali" tramite la combinazione lineare delle features originali, in modo da generare un nuovo spazio in cui le componenti principali sono proprio i vettori della nuova base ortonormale.

## 2.2 Il metodo

### 2.2.1 Dettagli matematici

In generale, dato un certo dataset  $X$  costituito da  $N$  campioni e  $p$  variabili (matrice  $N \times p$ ), come abbiamo visto nella Figura 1.2, possiamo costruire la matrice di varianza e covarianza come:

$$\Sigma = \frac{1}{N} X^T X$$

dove l'elemento  $\sigma_{ij} = \sigma_{ji}$  è il coefficiente di correlazione tra la variabile  $i$  e la variabile  $j$ , mentre l'elemento sulla diagonale  $\sigma_{ii}$  è la varianza della variabile  $i$ -esima. Per costruzione, la matrice è simmetrica e semidefinita positiva, quindi diagonalizzabile e con autovalori reali e non negativi.

Prima di procedere però, è necessario standardizzare i dati. Infatti, se le variabili appartengono a scale e domini molto diversi, quelle con la scala numerica più grande domineranno la varianza. Nel nostro caso, per esempio, oltre al fatto che vi sono alcune variabili binarie, la variabile Customer Value ha una scala molto più grande delle altre.

A questo punto, se vogliamo proiettare i nostri dati su uno spazio di dimensione  $k < p$ , utilizziamo un'applicazione lineare  $B^T$ , dove  $B \in \mathbb{R}^{p \times k}$ , in modo tale da massimizzare la varianza spiegata. Per "tornare indietro" allo spazio di dimensione  $p$ , basta riapplicare  $B$  ai dati proiettati.

Pertanto, dato un qualsiasi elemento del dataset originale  $x_i \in \mathbb{R}^p$ , la sua proiezione nel nuovo spazio di dimensione  $k$  è  $B^T x_i$ , mentre la sua ricostruzione nello spazio originale è  $\tilde{x}_i = BB^T x_i$ .

L'applicazione  $B$  che massimizza la varianza spiegata risolve un altro problema di ottimizzazione, ovvero la minimizzazione della seguente funzione di loss:

$$\mathcal{L}_B = \sum_{i=1}^N \|x_i - \tilde{x}_i\|_2^2$$

Le componenti principali che, come dicevamo prima, sono i vettori della base ortonormale del nuovo spazio, sono le direzioni lungo le quali si ha massima varianza, che corrispondono ai  $k$  autovalori più grandi della matrice di varianza e covarianza. La somma di questi autovalori viene detta varianza spiegata.

La matrice  $B$ , pertanto, è una semplice matrice di cambio di base, le cui colonne  $b_1, b_2, \dots, b_k$  sono le componenti principali ordinate in senso decrescente per varianza spiegata.

### 2.2.2 Ricerca delle componenti principali

La ricerca delle PC avviene in maniera progressiva. La prima componente viene ricavata risolvendo un problema di massimizzazione della varianza, mentre le altre vengono calcolate risolvendo lo stesso problema, ma con l'aggiunta del vincolo di ortogonalità rispetto a tutte le altre direzioni trovate precedentemente. Vediamolo nel dettaglio. Definiamo i seguenti elementi:

- $\mathbf{b}_i$ : i-esima componente principale;
- $\mathbf{z}_{in} = b_i^T x_n$ : è la proiezione del dato n-esimo nella componente i-esima;
- $\mathbf{Z}_i = (z_{i1}, z_{i2}, \dots, z_{in})$ : vettore delle proiezioni lungo la direzione i;
- $\mathbf{V}_i = Var(Z_i)$ : la varianza spiegata i-esima.

Pertanto, la varianza della prima componente è:

$$\begin{aligned}
 V_1 &= Var(Z_1) = \frac{1}{N} \sum_{n=1}^N z_{1n}^2 = \\
 &= \frac{1}{N} \sum_{n=1}^N (b_1^T x_n)^2 \\
 &= \frac{1}{N} \sum_{n=1}^N b_1^T x_n x_n^T b_1 \\
 &= b_1^T \left( \frac{1}{N} \sum_{n=1}^N x_n x_n^T \right) b_1
 \end{aligned}$$

dove all'interno della parentesi riconosciamo la matrice  $\Sigma$  di varianza e covarianza definita in precedenza, per cui possiamo scrivere:

$$V_1 = b_1^T \Sigma b_1$$

e passare al problema di massimizzazione <sup>1</sup>:

$$\begin{aligned} & \underset{b_1}{\text{maximize}} && b_1^T \Sigma b_1 \\ & \text{subject to} && \|b_1\|^2 = 1 \end{aligned} \tag{2.1}$$

Per risolvere il problema costruiamo la Lagrangiana:

$$\mathcal{L}_1(b_1, \lambda_1) = b_1^T \Sigma b_1 + (1 - b_1^T b_1) \lambda_1$$

Ponendo il gradiente pari a zero, per ricavare i punti critici che corrispondono alle soluzioni del nostro problema, si ottiene:

$$\begin{cases} \frac{\partial}{\partial b_1} \mathcal{L}_1 = \Sigma b_1 - \lambda_1 b_1 = 0 \\ \frac{\partial}{\partial \lambda_1} \mathcal{L}_1 = 1 - b_1^T b_1 = 0 \end{cases}$$

dove la seconda equazione corrisponde al vincolo da soddisfare, mentre la prima impone l'equazione agli autovalori:

$$\Sigma b_1 = \lambda_1 b_1$$

Moltiplicando tutto per  $b_1^T$  si ottiene che la varianza  $V_1$  è uguale proprio all'autovalore  $\lambda_1$  della matrice  $\Sigma$ , cioè quello maggiore (ricordiamo che gli autovalori sono tutti reali e non negativi).

Come detto precedentemente, per ricavare le altre componenti principali  $b_i$ , bisognerà aggiungere al problema di ottimizzazione visto per  $b_1$  dei vincoli che garantiscano l'ortogonalità tra le componenti. Pertanto, la componente principale  $i$ -esima sarà la soluzione del problema:

$$\begin{aligned} & \underset{b_i}{\text{maximize}} && b_i^T \Sigma b_i \\ & \text{subject to} && \|b_i\|^2 = 1, \\ & && b_i^T b_j = 0, \quad \forall j = 1, \dots, i-1 \end{aligned} \tag{2.2}$$

Alla fine di questo processo, se vogliamo ottenere uno spazio  $k$ -dimensionale, otterremo i vettori  $b_1, \dots, b_k$ , e la varianza totale spiegata sarà la somma

---

<sup>1</sup>Il vincolo imposto all'interno del sistema è cruciale, se non ci fosse, oltre al fatto che non staremmo costruendo una base ortonormale, la soluzione del problema di ottimizzazione sarebbe indefinita.

degli autovalori:

$$\text{Var}_{\text{spiegata}} = \sum_{i=1}^k \lambda_i$$

Equivalentemente, si potrebbe richiedere un minimo di varianza spiegata; in quel caso si troverebbero tutte le PC necessarie a tale scopo.

## 2.3 Applicazione al dataset

Abbiamo applicato la PCA al solo dataset originale. Infatti, come detto in precedenza, la PCA è un metodo di *unsupervised learning*, cioè non tiene conto della classificazione, dunque applicarla ai dati di test ricampionati non avrebbe alcun effetto. La prima cosa che abbiamo fatto è stata standardizzare i dati:

```

1 scaler = StandardScaler()
2 scaler.fit(X_train)
3 X_train_scaled= scaler.transform(X_train)
4
5 pca_X=PCA()
6 pca_X.fit(X_train_scaled)
```

In seguito abbiamo analizzato l'andamento della varianza spiegata all'aumentare del numero delle PC. Di seguito un grafico esplicativo.

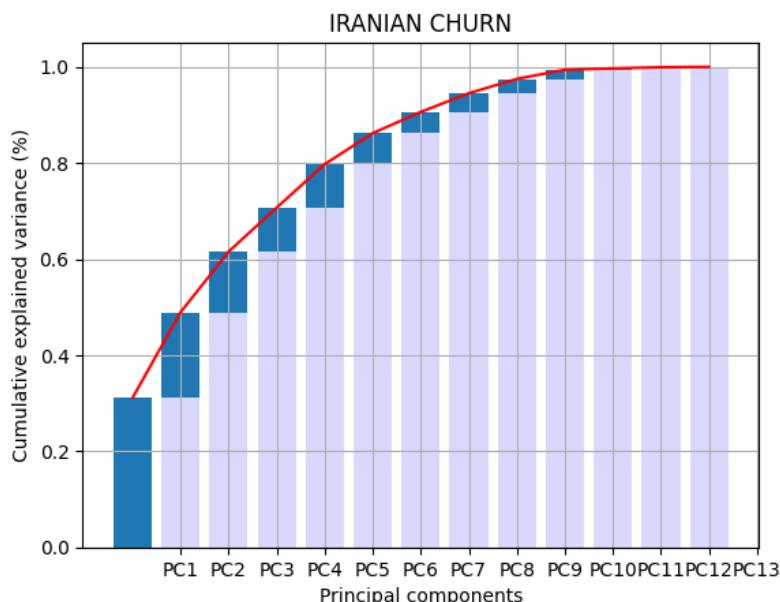


Figura 2.1: Andamento della varianza spiegata

Dall'immagine si può notare come, fino alla terza componente, la varianza spiegata sia leggermente superiore a quella del 60%. Ciò significa che, andando a visualizzare i dati nello spazio  $\mathbb{R}^3$  tramite, per esempio, uno Score Graph <sup>2</sup>, avremo comunque una perdita del 40% dell'informazione.

IRANIAN CHURN - SCORE GRAPH

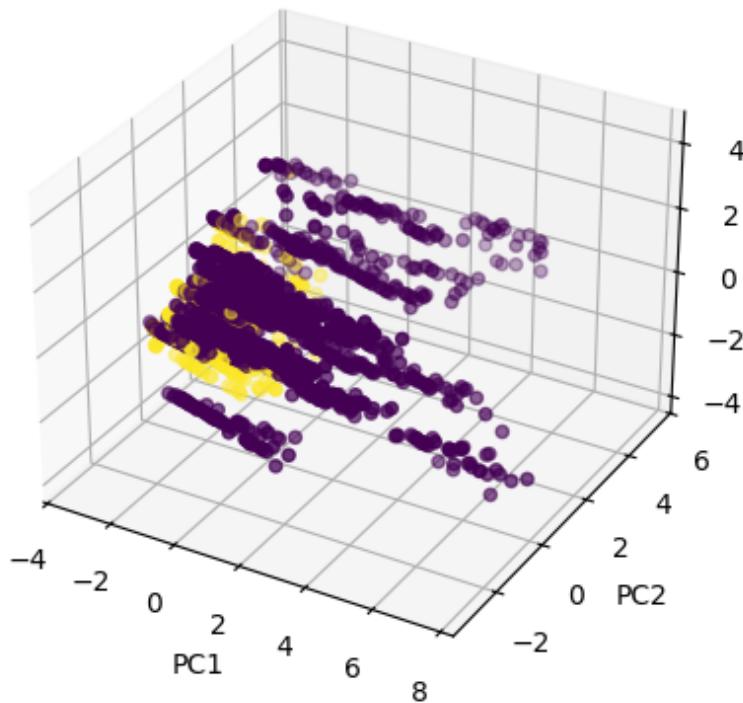


Figura 2.2: In viola la classe 0, in giallo la classe 1

Lavorando sempre con tre componenti, è possibile ottenere un grafico che ci permette di analizzare il dataset da un'altra prospettiva, ovvero quella delle PC.

Come spiegato, infatti, le componenti principali si ottengono tramite combinazione lineare delle variabili originali. Pertanto, può essere utile vedere quanto le vecchie features contribuiscono alla generazione delle nuove. Lo vediamo tramite un Loading Graph, che rappresenta per ogni variabile originale un vettore in  $\mathbb{R}^3$ . La direzione del vettore indica a quale componente esso contribuisce maggiormente, mentre la lunghezza esprime il contributo in termini di quantità. Inoltre, vettori vicini

---

<sup>2</sup>Grafico al cui interno vediamo la proiezione dei dati originali nel nuovo spazio di dimensione ridotta.

indicano variabili positivamente correlate.

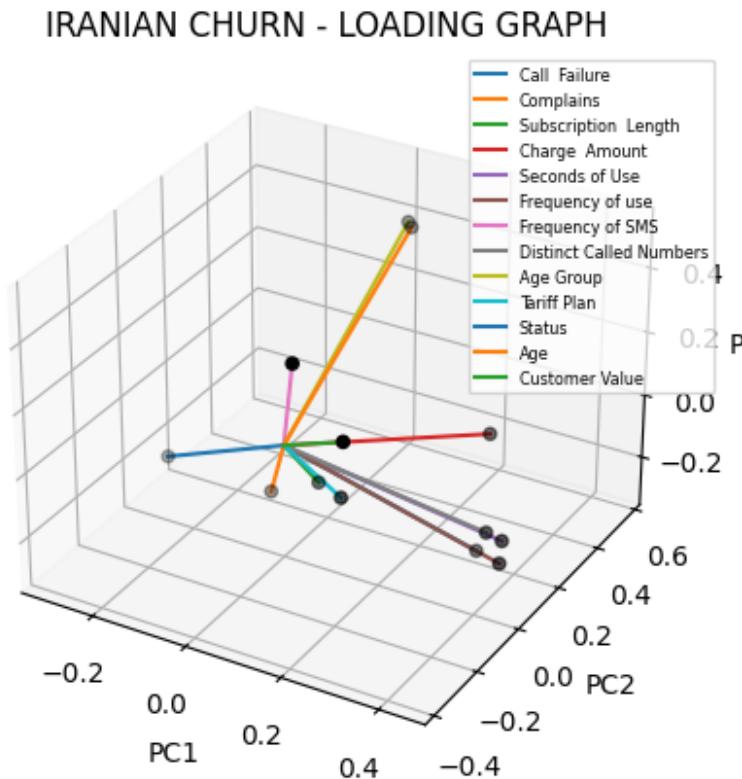
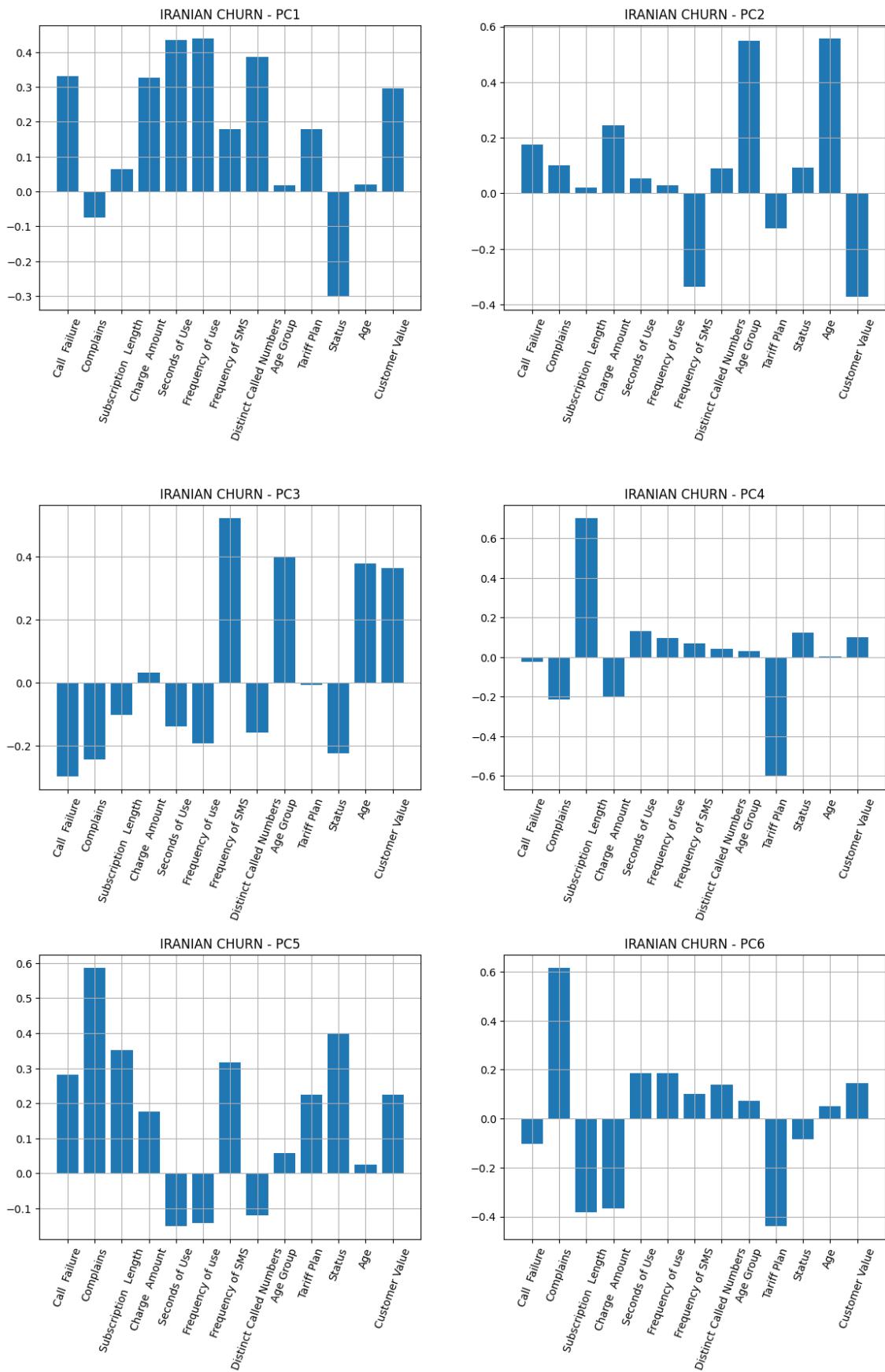


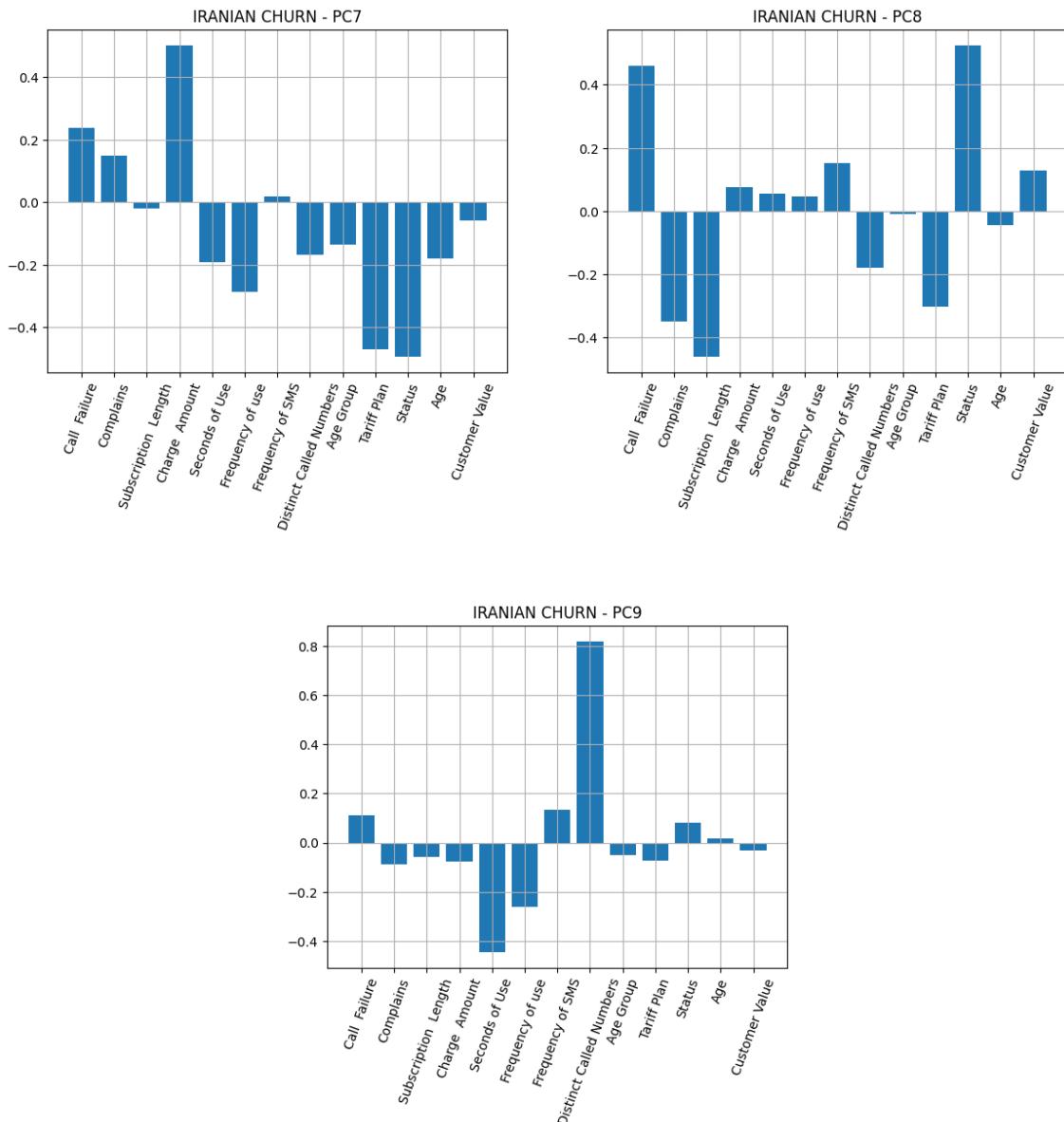
Figura 2.3: Loading Graph

Infine, guardando nuovamente alla figura 2.1, abbiamo osservato che, per ottenere una varianza spiegata totale del 95%, abbiamo bisogno di 9 componenti. Per questo, abbiamo generato ulteriori istogrammi che descrivono in maniera più intuitiva del Loading Graph quanto le features contribuiscono alle nuove PC.

## PCA

---







# Capitolo 3

## FDA

### 3.1 Introduzione

Un altro metodo per la proiezione dei dati in uno spazio di dimensione minore è la **Fisher Discriminant Analysis** (FDA). Tuttavia, rispetto alla PCA c'è un cambio totale di prospettiva. L'obiettivo della FDA, infatti, non è proiettare i dati minimizzando la perdita di informazione, bensì massimizzando la separazione tra le classi. A differenza della PCA, dunque, la Fisher Discriminant Analysis è un metodo di *supervised learning* proprio per questo motivo.

### 3.2 Il metodo

Vediamo un esempio per capire in cosa consiste la FDA.

Prendiamo, esattamente come nel nostro caso,  $N$  campioni  $x_i, i = 1, \dots, N$ , e supponiamo che essi siano separati in due classi. Abbiamo un totale di  $n_1$  campioni nella classe  $C_1$  e  $n_2$  campioni nella classe  $C_2$ . Come detto, dobbiamo proiettare questi dati in uno spazio di dimensione minore; per questo prendiamo una retta, la cui direzione è data dal versore  $v$ , e chiamiamo  $y_i = v^T x_i$  le proiezioni dei dati originali sulla retta.

Istintivamente, si potrebbe pensare che un modo molto efficiente per separare le due classi è cercare di massimizzare la distanza tra le medie dei dati proiettati  $|\tilde{\mu}_1 - \tilde{\mu}_2|$ , dove:

- 
- $\tilde{\mu}_1 = \frac{1}{n_1} \sum_{y_i \in C_1} y_i = v^T \mu_1$
  - $\tilde{\mu}_2 = \frac{1}{n_2} \sum_{y_i \in C_2} y_i = v^T \mu_2$

In realtà, spesso questo approccio non è molto efficiente, perché non tiene conto della varianza dei dati; per questo bisogna cercare di normalizzare la distanza tra le medie dei dati proiettati con qualcosa che sia proporzionale alla varianza.

Definiamo, dunque, per dei generici campioni  $z_i, i = 1, \dots, n$ , lo **scatter**:

$$s^2 = \sum_{i=1}^n (z_i - \mu_z)^2$$

Che non è altro che la varianza moltiplicata per la dimensione del campione, e rappresenta dunque un altro modo di misurare la dispersione dei dati. A questo punto definiamo le due quantità:

- $\tilde{s}_1^2 = \sum_{y_i \in C_1} (y_i - \tilde{\mu}_1)^2$
- $\tilde{s}_2^2 = \sum_{y_i \in C_2} (y_i - \tilde{\mu}_2)^2$

L'obiettivo della FDA è quello di trovare la direzione  $v$  che massimizza la quantità

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

Infatti, più è grande il numeratore, più le medie dei dati proiettati sono separate; più piccolo è il denominatore, più i dati proiettati sono vicini alla media della rispettiva classe.

Dobbiamo adesso riscrivere la funzione  $J$  cercando di esplicitare la dipendenza diretta dalla direzione  $v$ . Per questo si definiscono i seguenti elementi:

- $S_1 = \sum_{x_i \in C_1} (x_i - \mu_1)(x_i - \mu_1)^T$ : scatter matrix dei dati originali di classe 1;
- $S_2 = \sum_{x_i \in C_2} (x_i - \mu_2)(x_i - \mu_2)^T$ : scatter matrix dei dati originali di classe 2;

- $S_w = S_1 + S_2$ : **within scatter matrix**, misura la dispersione dei dati originali all'interno della classe di appartenenza;
- $S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$ : **between scatter matrix**, misura la separazione delle medie delle classi prima della proiezione.

Riscriviamo il denominatore in funzione di  $v$ , partendo da  $\tilde{s}_1^2$ .

$$\begin{aligned}\tilde{s}_1^2 &= \sum_{y_i \in C_1} (y_i - \tilde{\mu}_1)^2 \\ &= \sum_{x_i \in C_1} (v^T x_i - v^T \mu_1)^2 \\ &= \sum_{x_i \in C_1} (v^T(x_i - \mu_1))^T (v^T(x_i - \mu_1)) \\ &= \sum_{x_i \in C_1} v^T(x_i - \mu_1)(x_i - \mu_1)^T v \\ &= v^T S_1 v\end{aligned}$$

Analogamente, si ha  $\tilde{s}_2^2 = v^T S_2 v$ , per cui il denominatore è:

$$\tilde{s}_1^2 + \tilde{s}_2^2 = v^T S_1 v + v^T S_2 v = v^T S_w v$$

Per quanto concerne, invece, il numeratore, scriviamo:

$$\begin{aligned}(\tilde{\mu}_1 - \tilde{\mu}_2)^2 &= (v^T \mu_1 - v^T \mu_2)^2 \\ &= v^T(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T v \\ &= v^T S_B v\end{aligned}$$

Complessivamente:

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{v^T S_B v}{v^T S_w v}$$

Per ottenere il massimo di questa funzione, cerchiamone i punti critici derivando rispetto a  $v$ . Si ottiene l'equazione  $S_B v = \lambda S_w v$ , che è un problema agli autovalori generalizzato, dove  $\lambda = \frac{v^T S_B v}{v^T S_w v}$ .

Tuttavia, se la within scatter matrix ha rango pieno, ed è dunque invertibile, si ottiene un autentico problema agli autovalori:

$$S_w^{-1} S_B v = \lambda v$$

Che si risolve immediatamente ponendo  $v = S_w^{-1}(\mu_1 - \mu_2)$ , ricavando così la direzione cercata.

Chiaramente questo vale nel caso binario, di fronte a  $k$  classi si può generalizzare questo ragionamento e cercare le  $k-1$  direzioni individuate dalle corrispondenti equazioni agli autovalori.

### 3.3 Metodi di classificazione

#### 3.3.1 LDA

La **Linear Discriminant Analysis** è un metodo di classificazione che estende la teoria esposta precedentemente tramite la statistica bayesiana. L'ipotesi principale su cui si basa è che la distribuzione delle classi sia normale e che la matrice di varianza e covarianza sia la stessa tra le varie classi (omoschedasticità). Partendo dal teorema di Bayes sulla probabilità condizionata, esprimiamo la probabilità che, condizionatamente al dato  $x$ , la classe di appartenenza sia la classe  $k$ :

$$P(Y = k \mid X = x) = \frac{P(X = x \mid Y = k)P(Y = k)}{P(X = x)} = \frac{\pi_k f_k(x)}{\sum_{i=1}^{n_c} \pi_i f_i(x)}$$

dove:

- $\pi_k \in (0,1)$  corrisponde alla probabilità a priori di appartenere alla classe  $k$ ;
- $n_c$  è il numero di classi (nel nostro caso 2);
- $f_k(x) = \frac{1}{(2\pi)^{n/2}\sqrt{|S|}} e^{-\frac{1}{2}(x-\mu_k)^T S^{-1}(x-\mu_k)}$  è la funzione di densità della distribuzione normale.

Da questa formula si riesce a ricavare la funzione discriminante per ogni classe  $k$ :

$$\delta_k(x) = x^T S^{-1} \mu_k - \frac{1}{2} \mu_k^T S^{-1} \mu_k + \log \pi_k \quad \forall k = 1, \dots, n_c$$

dove  $S$  è la matrice di varianza e covarianza che, per l'ipotesi di omoschedasticità, è uguale per tutte le classi.

In generale, è ovvio che non siamo in possesso di tutti questi dati, pertanto andranno stimati. Denotiamo con  $\hat{\delta}_k$  la stima della funzione discriminante relativa alla k-esima classe, da cui possiamo approssimare la probabilità a cui eravamo interessati inizialmente:

$$\hat{P}(Y = k \mid X = x) = \frac{e^{\hat{\delta}_k(x)}}{\sum_{i=1}^{n_c} e^{\hat{\delta}_i(x)}}$$

### 3.3.2 QDA

Oltre alla LDA, un altro metodo di classificazione è la **Quadratic Discriminant Analysis** (QDA). Questo metodo lavora in maniera analoga, supponendo che la distribuzione delle classi sia normale ma, a differenza della LDA, assumendo che le matrici di varianza e covarianza delle classi siano diverse (eteroschedasticità). Un'altra differenza sostanziale, che si evince anche da come questi metodi sono chiamati, è che con la LDA separiamo le classi tramite iperpiani (Linear DA), mentre con la QDA troviamo delle curve di separazione (Quadratic DA).

In maniera molto simile a quanto fatto prima, si trova la seguente funzione discriminante:

$$\delta_k(x) = -\frac{1}{2}(x - \mu_k)^T S_k^{-1}(x - \mu_k) + \log \pi_k - \frac{1}{2} \log |S_k| \quad \forall k = 1, \dots, n_c$$

Anche in questo caso tutti i parametri dovranno essere stimati, quindi avremo una funzione discriminante approssimata  $\hat{\delta}_k$ . Notiamo che, sotto l'ipotesi di eteroschedasticità, il numero di parametri da stimare è maggiore rispetto a quello della LDA, proprio perché le matrici di covarianza sono diverse.

Tutto ciò vuol dire che, in generale, la QDA lavora meglio rispetto alla LDA quando le covarianze tra le classi sono realmente diverse, e quando si hanno dati a sufficienza per poterle stimare correttamente.

## 3.4 Applicazione al dataset

Come già discusso ampiamente durante l'analisi del dataset, molte features sono binarie o hanno comunque domini discreti, pertanto sarebbe

molto ambizioso aspettarsi che i dati delle classi abbiano distribuzione normale. Per questo, è inutile applicare i metodi di classificazione appena descritti, ma può essere interessante provare comunque a proiettare i dati tramite la FDA, cercando la massima separazione tra le classi. Essendoci soltanto due classi, i dati vengono proiettati su una retta, con lo scopo di massimizzare la distanza tra le medie e minimizzare la varianza "interna" (espressa dalla within scatter matrix).

Ecco i risultati ottenuti:

```

1 fda = MDA()
2 y_train=y_train.iloc[:, 0]
3 fda.fit(X_train.values,y_train.values)
4 Zfda = fda.transform(X_train.values)
5 fig1, axs1 = plt.subplots(figsize=(7, 5))
6 axs1.scatter(Zfda[:, 0], [0]*len(Zfda), c=y_train, alpha=0.5)
7 axs1.set_title('FDA')
8 fig1.savefig("FDA.png", bbox_inches='tight')
9 axs1.grid()

```

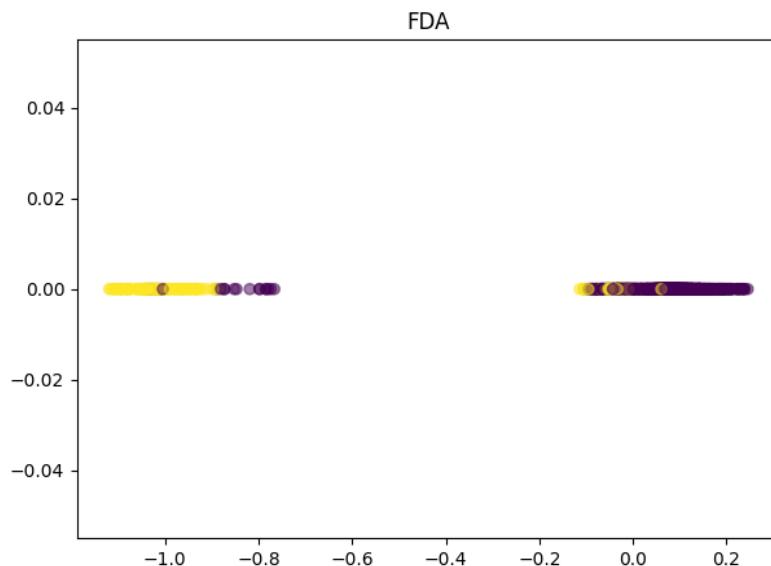


Figura 3.1: Proiezione con FDA

# Capitolo 4

## SVM

### 4.1 SVM lineari

Le SVM (*Support Vector Machines*) sono un particolare algoritmo di apprendimento supervisionato che ha come obiettivo quello di trovare l'iperpiano migliore in grado di separare due classi in uno spazio n-dimensionale.

#### 4.1.1 Dettagli matematici

##### Hard SVM

Consideriamo un *dataset*  $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$  dove  $m = |S|$ ,  $\mathbf{x}_i \in \mathbb{R}^d$  e  $y_i \in \{1, -1\} \forall i = 1, \dots, m$ , in cui le variabili  $\mathbf{x}_i$  sono divise in due classi. Le  $y_i$  rappresentano infatti un'etichetta che diamo a tali valori

$$y_i = \begin{cases} 1 & \text{se } \mathbf{x}_i \in \text{Classe 1} \\ -1 & \text{se } \mathbf{x}_i \in \text{Classe 2} \end{cases} \quad (4.1)$$

Se i dati sono linearmente separabili l'idea è quella di dividerli utilizzando un iperpiano n-dimensionale massimizzando l'ampiezza del margine di separazione.

Definiamo quindi con  $\Pi_{w,b} := \{\mathbf{x} \in \mathbb{R}^n \mid \boldsymbol{\omega}^T \mathbf{x} + b = 0\}$  il generico iperpiano n-dimensionale, dove  $\boldsymbol{\omega} \in \mathbb{R}^n$  è il vettore normale e  $b \in \mathbb{R}$  è un parametro, e con  $\text{dist}(\Pi_{w,b}, \mathbf{x})$  la distanza euclidea di tale iperpiano da

un generico vettore  $\mathbf{x} \in \mathbb{R}^n$ . Ricordiamo che la formula della distanza euclidea è data da:

$$\text{dist}(\Pi_{w,b}, \mathbf{x}) = \frac{|\boldsymbol{\omega}^T \mathbf{x} + b|}{\|\boldsymbol{\omega}\|} \quad (4.2)$$

Il margine, cioè la minima distanza tra i dati e l'iperpiano sarà definito in questo modo:

$$M_{\boldsymbol{\omega}, b} := 2 \cdot \min_{\mathbf{x} \in S} (\text{dist}(\Pi_{w,b}, \mathbf{x})) \quad (4.3)$$

Un vettore  $x_i$  viene chiamato *support vector* se si trova sul margine. I *support vector* sono i punti più importanti perché sono quelli che determinano maggiormente l'iperpiano separatore.

Date queste informazioni preliminari il problema da risolvere è il seguente:

$$\arg \max_{(\boldsymbol{\omega}, b) : \|\boldsymbol{\omega}\| = 1} \min_{i \in [m]} |\langle \boldsymbol{\omega}, \mathbf{x}_i \rangle + b| \quad (4.4a)$$

$$\text{subject to } y_i(\langle \boldsymbol{\omega}, \mathbf{x}_i \rangle + b) > 0, \quad \forall i = 1, \dots, m \quad (4.4b)$$

Come si può notare, stiamo cercando di massimizzare il minimo della distanza tra l'iperpiano e i punti del dataset (cerchiamo di ottenere il massimo margine possibile) e stiamo imponendo il vincolo di separabilità tra le due classi. L'iperpiano viene scelto direttamente con  $\|\boldsymbol{\omega}\| = 1$  così da non avere problemi al denominatore della funzione obiettivo. Tale problema di ottimizzazione può essere riscritto in questo modo:

$$\arg \min \frac{1}{2} \|\boldsymbol{\omega}\|^2 \quad (4.5a)$$

$$\text{subject to } y_i(\langle \boldsymbol{\omega}, \mathbf{x}_i \rangle + b) \geq 1, \quad \forall i = 1, \dots, m \quad (4.5b)$$

Dove  $\frac{1}{2}$  è stato inserito per comodità di calcolo. Ora possiamo applicare il metodo dei moltiplicatori di Lagrange introducendo le variabili  $\alpha_i$  e imporre le condizioni KKT trovando la formulazione duale del problema:

$$\begin{aligned}
 & \underset{\alpha}{\text{maximize}} && \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\
 & \text{subject to} && \sum_i \alpha_i y_i = 0, \quad \forall i = 1, \dots, m, \\
 & && \alpha_i \geq 0, \quad \forall i = 1, \dots, m
 \end{aligned} \tag{4.6}$$

con

$$\alpha_i \begin{cases} = 0 & \text{se } \mathbf{x}_i \text{ non è support vector} \\ \neq 0 & \text{se } \mathbf{x}_i \text{ è support vector} \end{cases} \tag{4.7}$$

troviamo quindi la soluzione

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^m \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b\right) \tag{4.8}$$

dove:

$$b = y_i - \sum_{j=1}^m \alpha_j y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \quad \forall \mathbf{x}_i \text{ Support Vector} \tag{4.9}$$

Come possiamo notare la soluzione dipende esclusivamente dai *support vector* e non dai punti più distanti.

## Soft SVM

Spesso è difficile separare perfettamente i dati con un iperpiano. Se si vuole tuttavia mantenere una separazione lineare può essere utile rilassare il vincolo sul margine. Se nella *Hard* SVM non era ammesso che un punto potesse superare il margine, nella *Soft* SVM tale possibilità è contemplata.

Si introducono quindi delle variabili decisionali ausiliarie, le  $\xi_i$ , variabili di *slack* che rappresentano il "prezzo che pago" ad attraversare il margine, e  $C$ , una costante che svolge lo ruolo di ammortizzatore: indica quanto sono interessato ad attraversare il confine. Si ottiene quindi un problema formulato nel seguente modo:

$$\arg \min \frac{1}{2} \|\boldsymbol{\omega}\|^2 + C \sum_{i=1}^m \xi_i \quad (4.10a)$$

$$\text{subject to } y_i(\langle \boldsymbol{\omega}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad \forall i = 1, \dots, m, \quad (4.10b)$$

$$\xi_i \geq 0, \quad \forall i = 1, \dots, m \quad (4.10c)$$

Applicando anche in questo caso il metodo dei moltiplicatori di Lagrange e imponendo le condizioni KKT si ottiene la formulazione duale del problema:

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} \quad \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ & \text{subject to} \quad \sum_i^m \alpha_i y_i = 0, \quad \forall i = 1, \dots, m, \\ & \quad \alpha_i \geq 0, \quad \forall i = 1, \dots, m, \\ & \quad \alpha_i \leq C, \quad \forall i = 1, \dots, m \end{aligned} \quad (4.11)$$

con soluzione

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^m \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b\right) \quad (4.12)$$

dove:

$$b = y_i - \sum_{j=1}^m \alpha_j y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \quad \forall \mathbf{x}_i \text{ Support Vector} \quad (4.13)$$

Notiamo che la soluzione è la medesima del problema precedente e l'unica differenza sta nei vincoli del problema: le  $\alpha_i$  ora hanno come dominio di definizione l'intervallo  $[0, C]$ . Dalla costante  $C$  dipende quindi il comportamento della soluzione:

- se  $C \rightarrow 0$  aumenta la "morbidezza" del margine: le variabili  $\xi_i$  conteranno poco all'interno della funzione obiettivo e le variabili  $\alpha_i$  saranno anch'esse prossime allo 0 (nessun punto viene riconosciuto come *support vector*) e quindi i *support vector* non verranno distinti dai punti lontani dal margine.
- se  $C \rightarrow \infty$  il margine sarà duro e si torna al caso delle *hard SVM*: le

$\xi_i$  conteranno molto nella funzione obiettivo, una minima trasgressione nel superamento del margine verrà fatta pagare parecchio, e le  $\alpha_i$  invece avranno come dominio un intervallo illimitato a destra come nel caso precedente.

#### 4.1.2 Applicazione al dataset

Le SVM lineari addestrate fanno riferimento ai dati originali con un netto sbilanciamento per la classe 0, ai dati bilanciati artificialmente con il metodo ROS (*Random Over Sampler*) e ai dati bilanciati attraverso il metodo SMOTE (*Synthetic Minority Over-sampling Technique*). Tutte le SVM sono state effettuate sui dati nella loro dimensione naturale (14 *features*) e poi dopo aver applicato una PCA con 2 come numero di *features*. Questo per avere una interpretazione visiva a costo di una peggiore accuratezza. Lo scopo della proiezione è solo didattico: vedere come si comporta una SVM in presenza di una possibile predisposizione dei dati.

In tutti i casi i dati sono stati divisi in un *training set* del 65% e un *test set* del 35%.

Ho inizializzato una SVM lineare utilizzando come parametro  $C$  un numero molto elevato per imitare un margine duro.

```

1 # applico la SVM ai dati con la PCA e senza la PCA
2 C_hard = 1e10
3 loss = 'squared_hinge'
4 dual = False
5 random_seed = 42
6
7 lsvm_hard = LinearSVC(C=C_hard, loss=loss, dual=dual, random_state=
8     random_seed)
9 lsvm_hard_pca = LinearSVC(C=C_hard, loss=loss, dual=dual,
10    random_state=random_seed)
11
12 lsvm_hard.fit(X_train, y_train)
13 lsvm_hard_pca.fit(pca.transform(X_train), y_train)

```

I risultati ottenuti sono schematizzati nella seguente tabella che ho ottenuto grazie al metodo *classification\_report()*:

Tabella 4.1: SVM lineare per i dati non ricampionati

Label	Precision	Recall	F1-score	Support
0	0.90	0.99	0.94	930
1	0.86	0.41	0.55	173
<b>Accuracy</b>			0.90	1103
<b>Macro avg</b>	0.88	0.70	0.75	1103
<b>Weighted avg</b>	0.89	0.90	0.88	1103

In questa tabella vengono analizzati alcuni parametri importanti per stabilire la buona riuscita di un modello. Essi assumono valori nell'intervallo [0,1]. La *precision* indica la precisione del modello nel predire una determinata classe  $C_i$  ed è definita nel seguente modo:

$$\text{Precision}(C_i) = \frac{\text{Veri } C_i}{\text{Falsi } C_i + \text{Veri } C_i} \quad (4.14)$$

La *recall* indica l'abilità del modello nel trovare tutti i campioni della classe  $C_i$  nel *test set*  $TS$ :

$$\text{Recall}(C_i) = \frac{\text{Veri } C_i}{\# \text{ di } C_i \text{ in TS}} \quad (4.15)$$

Infine l'*F1-score* è una misura che aggrega le informazioni contenute nella *precision* e nella *recall*. Esso viene più in generale definito come *F $\beta$ -score* in questo modo:

$$F_\beta(C_i) = (1 + \beta^2) \frac{\text{Precision}(C_i) \cdot \text{Recall}(C_i)}{\beta^2 \text{Precision}(C_i) + \text{Recall}(C_i)} \quad (4.16)$$

Nel nostro caso con  $\beta = 1$  diamo la stessa importanza a *precision* e *recall*.

Notiamo che la classe 0 maggioritaria viene indovinata con grande esattezza mentre per la classe 1 le prestazioni sono molto basse: si ha una *recall* di appena 0.41, meno della metà della classe uno viene indovinata.

Applicando la PCA addestrata ai dati e stampando il grafico con i punti e l'iperpiano separatore si ottiene questo:

```

1 # Pesi retta (iperpiano di R^2) separatrice
2 w_hard = lsvm_hard_pca.coef_

```

```

3 b_hard = lsvm_hard_pca.intercept_
4
5 # Inizializzazione retta separatrice
6 line_hard = HyperplaneR2(w_hard, b_hard)
7
8 Zpca = pca.transform(X_test)
9
10 plt.figure()
11 plt.scatter(Zpca[:, 0], Zpca[:, 1], c=y_test, alpha=0.05)
12 plt.plot([0., 1000.], [line_hard.line_x2(0.), line_hard.line_x2(1000.)], label='sep. hyperplane (svm)')
13 plt.plot([0., 1000.], [line_hard.margin_x2(0.)[0], line_hard.margin_x2(1000.)[0]], 'r--', label='margin border')
14 plt.plot([0., 1000.], [line_hard.margin_x2(0.)[1], line_hard.margin_x2(1000.)[1]], 'r--')
15 plt.xlabel('PC1')
16 plt.ylabel('PC2')
17 plt.grid()
18 plt.title('2-PC Visualization (score graph)')
19 plt.savefig('SVM_lineare_2.png', dpi=300, bbox_inches='tight')

```

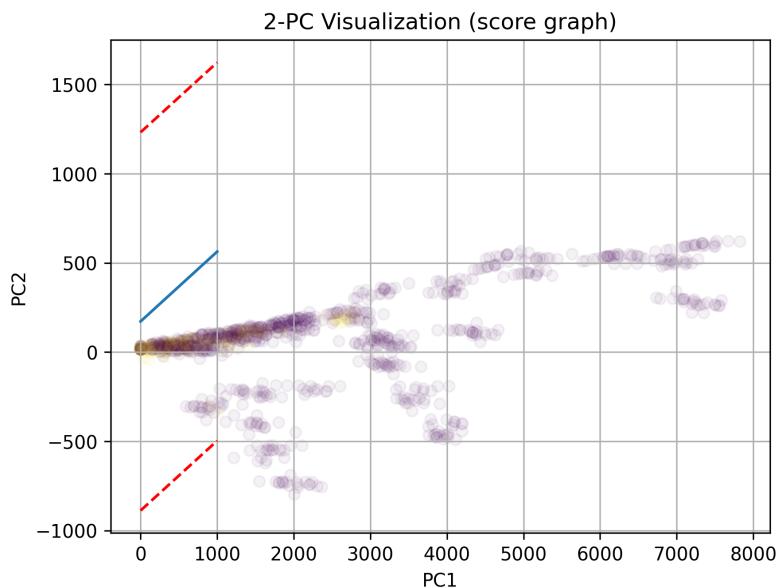


Figura 4.1: SVM lineare dataset iniziale

Come si può notare la retta separatrice cerca di classificare correttamente tutti i punti in viola appartenenti alla classe maggioritaria e non classifica correttamente nemmeno un punto della classe minoritaria, questo è un problema palusibile se il *dataset* è sbilanciato.

Per ciò che concerne i dati ricampionati con ROS ho creato i *set* di addestramento modificati e ho lasciato inalterati i dati di *test* per evitare *data leak*, ho quindi applicato gli stessi comandi precedentemente elencati ottenendo i seguenti risultati:

Tabella 4.2: SVM lineare per i dati ricampionati con ROS

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.98	0.82	0.89	930
1	0.48	0.90	0.62	173
<b>Accuracy</b>			0.83	1103
<b>Macro avg</b>	0.73	0.86	0.76	1103
<b>Weighted avg</b>	0.90	0.83	0.85	1103

L'accuratezza generale si è abbassata (da 0.9 a 0.83) e anche i valori riguardanti le predizioni della classe maggioritaria, ad esempio la *recall* scende da 0.99 a 0.82. La classe minoritaria ha avuto un incremento nella *recall* (da 0.41 a 0.9) e un netto decremento nella *precision*. Questi risultati, che a prima vista possono sembrare di cattivo auspicio, sono in realtà indice di un miglioramento delle prestazioni. Il nostro modello si è "accorto" della presenza della classe minoritaria e ha cercato di includerla nella classificazione abbassando le prestazioni per la classe rivale. Questo fatto si può notare nella proiezione bidimensionale dei dati:

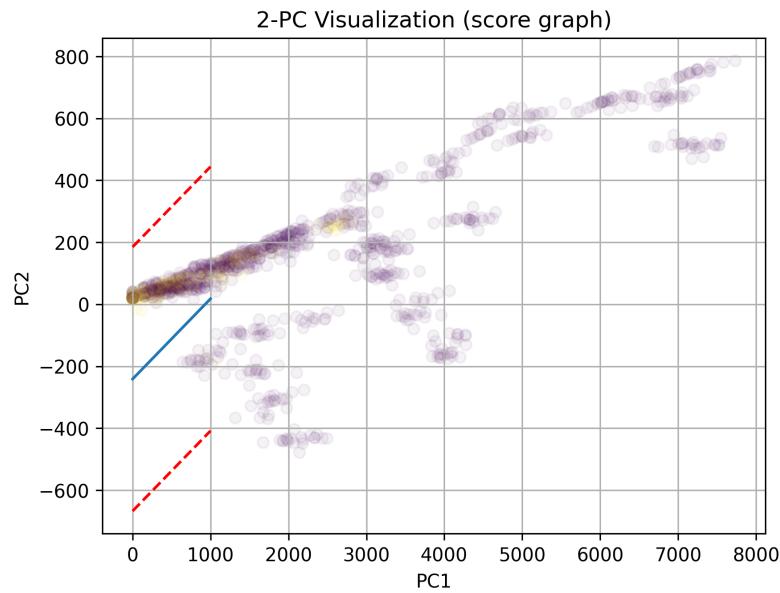


Figura 4.2: SVM lineare dataset ricampionato con ROS

Adesso la retta separatrice cerca di dividere entrambe le classi mettendo i punti in giallo alla sua sinistra e quelli in viola alla sua destra, un buon segno.

Infine, per quanto riguarda i dati ricampionati con SMOTE si ottengono i reguenti risultati:

Tabella 4.3: SVM lineare per i dati ricampionati con SMOTE

Label	Precision	Recall	F1-score	Support
0	0.97	0.83	0.90	930
1	0.50	0.87	0.63	173
<b>Accuracy</b>			0.84	1103
<b>Macro avg</b>	0.73	0.85	0.76	1103
<b>Weighted avg</b>	0.90	0.84	0.86	1103

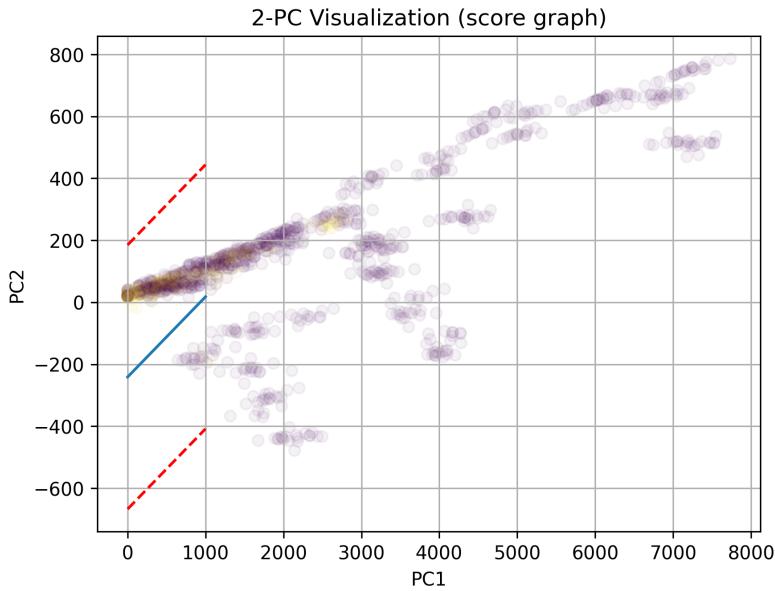


Figura 4.3: SVM lineare dataset ricampionato con SMOTE

I valori delle metriche rimangono pressoché invariati con i dati campionati con ROS, i due metodi in questo caso hanno portato a risultati molto simili.

## 4.2 SVM non lineari

### 4.2.1 Dettagli matematici

Molto spesso i dati non sono linearmente separabili (nemmeno se rilassiamo il vincolo sul margine). Essi infatti possono essere disposti nei modi più disparati possibili, secondo le forme più bizzarre. In questi casi si possono utilizzare le SVM non lineari.

Le SVM non lineari si fondano su un principio geometrico: più le dimensioni in cui si trovano i dati sono elevate più è facile trovare un iperpiano che li separa. Una prova intuitiva di questo fatto è, ad esempio, la seguente.

Immaginiamo due cerchi concentrici in  $\mathbb{R}^2$  essi non sono in alcun modo separabili linearmente. Proviamo ora a mapparli in uno spazio di dimensione maggiore come  $\mathbb{R}^3$  attraverso la seguente trasformazione:

$$(x, y) \longrightarrow (x, y, x^2 + y^2)$$

In questo modo essi sono separabili con un piano perchè si trovano ad altezze diverse sull'asse  $z$ .

L'idea è quindi quella di spostare i dati in una dimensione maggiore rispetto a quella in cui si trovano tramite una trasformazione adatta, separarli linearmente in questo spazio e poi riapplicare la trasformazione inversa. Più formalmente si definisce una mappa  $\Phi : X \rightarrow \hat{X}$  con  $X \subset \mathbb{R}^d$  e  $\hat{X} \subset \mathbb{R}^e$  con  $e > d$  e si trova l'iperpiano separatore con equazione

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^m \alpha_i y_i \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle_{\mathbb{R}^e} + b\right) \quad (4.17)$$

dove

$$b = y_i - \sum_{j=1}^m \alpha_j y_j \langle \Phi(\mathbf{x}_j), \Phi(\mathbf{x}_i) \rangle_{\mathbb{R}^e}, \quad \forall \mathbf{x}_i \text{ Support Vector} \quad (4.18)$$

Sorgono però dei problemi:

- trasformare i dati con  $\Phi$  nel modo sopra descritto risulta costoso computazionalmente in quanto i *dataset* sono molto grandi
- i dati solitamente hanno molte caratteristiche e vivono già in spazi con dimensioni alte, aumentarle ulteriormente può risultare inefficiente
- spesso si può andare in contro al fenomeno dell'*overfitting*

Per ovviare ai primi due problemi si fa ricorso al *Kernel Trick*. Notiamo infatti che l'iperpiano separatore dipende solamente dalle valutazioni dei prodotti scalari tra variabili trasformate tramite le  $\Phi$  e non dai valori esplicativi dei dati trasformati. Mappare i dati è quindi un passaggio superfluo e inutilmente dispendioso, basta definire un *Kernel* che riproduca un prodotto scalare:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \quad (4.19)$$

Non tutti i *Kernel* che si possono definire sono in grado di indurre un prodotto scalare nello spazio in cui sono definiti, essi devono rispettare certe condizioni. Il seguente lemma ne è un esempio.

**Lemma 1 (Condizioni di Mercer)** *Sia  $K : X \times X \rightarrow \mathbb{R}$  un nucleo simmetrico. Tale nucleo è in grado di generare un prodotto scalare in uno spazio  $H$  di Hilbert  $\Leftrightarrow \forall n \leq m, \forall \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset X$  la matrice  $(G_{ij}) = K(\mathbf{x}_i, \mathbf{x}_j)$  è semidefinita positiva*

I *Kernel* oggi implementati sono standard e si conosce molto bene il loro comportamento. Ogni *Kernel* ha il proprio ambito di utilizzo ed è specifico per il problema che si deve affrontare.

#### 4.2.2 Applicazione al dataset

Anche per le SVM non lineari sono stati utilizzati gli stessi dati, le stesse tecniche di campionamento, le stesse percentuali per la divisione in addestramento e inferenzia e le stesse PCA che per le SVM lineari.

Il codice che ho utilizzato per inizializzare le SVM non lineari è il seguente:

```

1 random_state = 42
2 ker_rbf = 'rbf'
3 gamma_rbf = 'auto'
4
5 C1 = 5
6 C2 = 1e2
7
8 # Inizializzazione SVM
9 svm_rbf_1 = SVC(C=C1, kernel=ker_rbf, gamma=gamma_rbf, random_state
10      =random_state)
11 svm_rbf_2 = SVC(C=C2, kernel=ker_rbf, gamma=gamma_rbf, random_state
12      =random_state)
13
14 # Addestramento SVM
15 svm_rbf_1.fit(X_train, y_train)
16 svm_rbf_2.fit(X_train, y_train)

```

Ho deciso di utilizzare il *kernel* RBF che induce

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}} \quad (4.20)$$

come prodotto scalare e due parametri di  $C$  per durezze del margine diverse.

A seguire i risultati ottenuti per tutti e tre i *dataset* (sono stati scelti i risultati migliori per ogni caso):

Tabella 4.4: SVM non lineare per i dati non ricampionati margine  $C1$

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.87	0.99	0.93	930
1	0.84	0.24	0.37	173
<b>Accuracy</b>			0.87	1103
<b>Macro avg</b>	0.86	0.61	0.65	1103
<b>Weighted avg</b>	0.87	0.87	0.84	1103

Tabella 4.5: SVM non lineare per i dati ricampionati con ROS margine  $C2$

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.88	0.98	0.93	930
1	0.76	0.25	0.38	173
<b>Accuracy</b>			0.87	1103
<b>Macro avg</b>	0.82	0.62	0.65	1103
<b>Weighted avg</b>	0.86	0.87	0.84	1103

Tabella 4.6: SVM non lineare per i dati ricampionati con SMOTE margine  $C2$

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.88	0.98	0.93	930
1	0.76	0.26	0.39	173
<b>Accuracy</b>			0.87	1103
<b>Macro avg</b>	0.82	0.62	0.66	1103
<b>Weighted avg</b>	0.86	0.87	0.84	1103

Le prestazioni come prima sono molto elevate per la classe maggioritaria mentre calano drasticamente per quella con una presenza minore. Notiamo infatti che la *recall* è molto bassa: ciò sta a indicare che si classificano troppi falsi negativi per la classe 1. In questo modo vengono classificate come persone che riangono con la compagnia individui che invece la abbandonano: ciò non è ottimale.

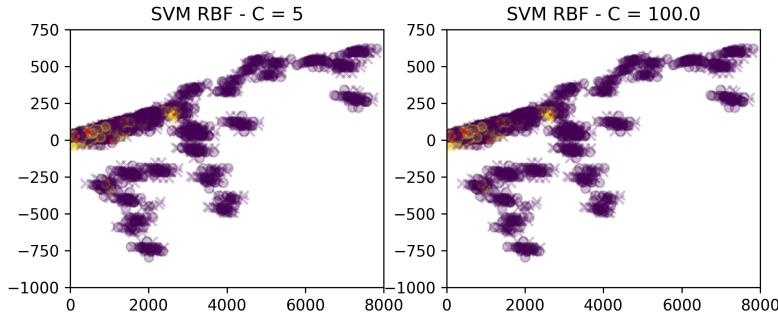


Figura 4.4: SVM non lineare dataset originale

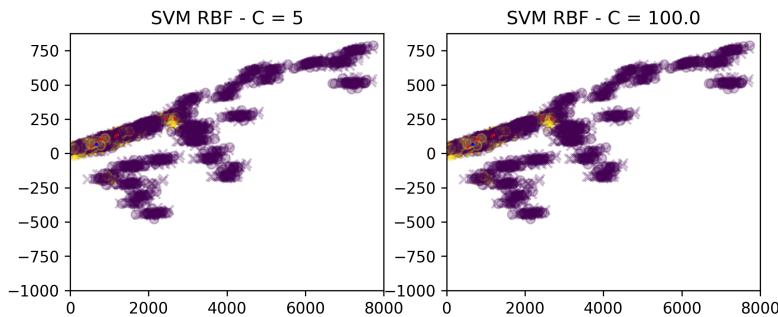


Figura 4.5: SVM non lineare dataset ricampionato con ROS

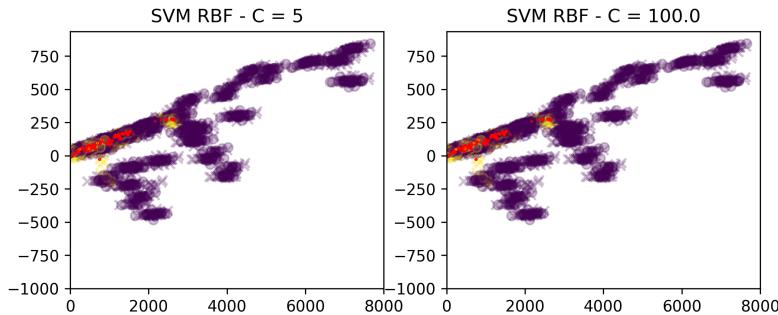


Figura 4.6: SVM non lineare dataset ricampionato con SMOTE

Dai grafici possiamo evincere che il kernel che utilizziamo sta cercando di creare delle "pozzanghere" nelle quali racchiudere i valori in giallo. Per separare i dati proiettati nelle due dimensioni potrebbe essere una strategia funzionante in quanto essi non sono linearmente separabili. Per ciò che concerne i dati nelle dimensioni originali non possiamo sapere se questa strategia è la migliore o se si possono incrementare le prestazioni in quanto non possiamo rappresentare graficamente le informazioni.

# Capitolo 5

## MLP

Un MLP (*Multilayer Perceptron*, Percettrone Multistrato) è un modello di rete neurale artificiale che, presi dei dati input, li restituisce in output in modo appropriato.

### 5.1 Dettagli matematici

I *Multilayer perceptron* poggiano la loro esistenza su di un teorema: il teorema di Kolmogorov, qui di seguito riportato.

**Teorema 1 (di Kolmogorov)** *Ogni funzione continua  $g : [0,1]^p \rightarrow \mathbb{R}$  con  $p \geq 2$  può essere riscritta come:*

$$g(\mathbf{x}) = \sum_{j=1}^{2p+1} h_j \left( \sum_{i=1}^p h_{ij}(x_i) \right) \quad (5.1)$$

con  $h_j$  e  $h_{ij}$  funzioni univariate continue  $\forall i \leq p, j \leq 2p + 1$

Questo teorema afferma che ogni funzione multivariata che sia almeno continua è in realtà scrivibile come composizione di funzioni univariate. Grazie a questo risultato possiamo abbozzare una rete neurale, come riportato nell'immagine qui sotto. La struttura è quella di un grafo completamente connesso composto da degli strati (*layers*) formati dalle  $z_j = \sum_{i=1}^p h_{ij}$  e dalle  $a_j = h_j(z_j) \quad \forall i, j$ .

Il teorema di Kolmogorov tuttavia stabilisce solo l'esistenza delle  $h_i$  e delle  $h_{ij}$ , nella dimostrazione non è contenuta una "ricetta" per trovarle. Abbiamo quindi  $(2p + 1)(p + 1)$  funzioni incognite; per risolvere questo

problema l'idea è quella di sostituire tale numero di funzioni sconosciute con un numero maggiore di funzioni note, chiamate funzioni di attivazione. La speranza è che una rete costruita in questo modo abbia lo stesso potere approssimante della rete di Kolmogorov.

Le funzioni di attivazione sono semplici per poter garantire un *learner* con una discreta capacità di rappresentazione a basso costo computazionale. Tali funzioni fanno inoltre in modo che una rete neurale non sia una semplice regressione lineare ma sia anche in grado di poter gestire eventuali non linearità. Un esempio di funzione di attivazione (che utilizzeremo nel nostro MLP) è la ReLU, definita in questo modo:

$$\text{ReLU}(x) = \max(0, x) \quad (5.2)$$

Un altro modo per migliorare le capacità rappresentative di una rete neurale è quello di aumentare gli strati (*hidden layers*).

Alla luce dei risultati qui sopra riportati le variabili una rete neurale con  $L+1$  *hidden layers* può essere schematizzata definendo  $z_l = W_l a_{l-1} + b_l$  e  $a_l = S_l(z_l) \quad \forall l \in \{0, \dots, L\}$  con  $W_l$  matrice dei pesi,  $b_l$  vettore dei bias,  $S_l$  funzione di attivazione e scrivendo:

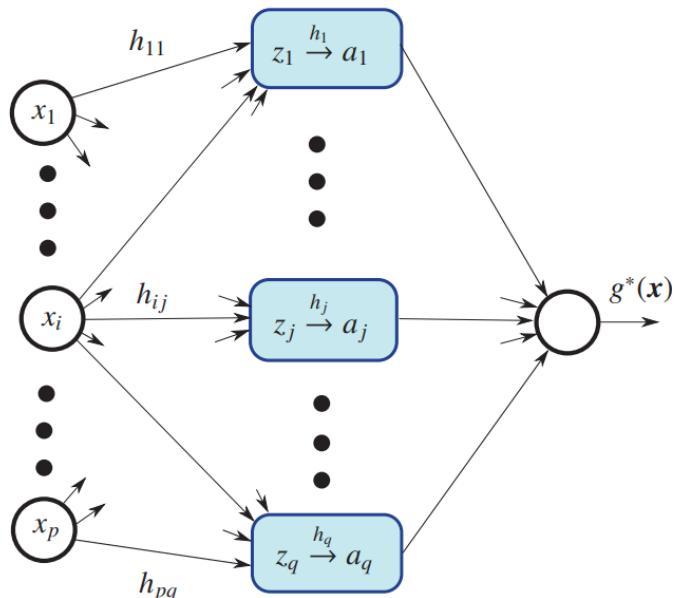


Figura 5.1: Abbozzo di un MLP

$$\begin{aligned} \mathbf{x} = a_0 &\rightarrow W_1 a_0 + b_1 = z_1 \rightarrow S_1(z_1) = a_1 \rightarrow \dots \\ &\rightarrow W_L a_{L-1} + b_L = z_L \rightarrow S_L(z_L) = a_L = g(\mathbf{x}) \end{aligned} \quad (5.3)$$

I parametri che popolano le  $W_l$  e i vettori  $b_l$  sono ignoti e vengono trovati durante la fase di addestramento della rete.

Per poter addestrare una rete e scovare chi sono i migliori parametri da inserire è necessario definire una quantità in grado di misurare l'efficacia nel restituire un output corretto. Tale quantità è la *loss*. La *loss* misura quanto l'algoritmo sbaglia nel dare una risposta. Vi sono diverse *loss functions*, quella più comune è il *Mean Squared Error* (errore quadratico medio). Nei problemi di classificazione si usa invece solitamente la *log-loss*.

Definita una *loss* adatta, addestrare una rete equivale a trovare la soluzione al problema

$$\underset{\omega}{\text{minimize}} \quad \text{Loss}(\omega) \quad (5.4)$$

dove speriamo di ottenere  $\omega^*$  tale per cui  $\text{Loss}(\omega^*) = 0$ .

## 5.2 Applicazione al dataset

Anche in questo caso abbiamo applicato l'MLP ai tre dataset descritti in precedenza.

L'MLP è stato inizializzato nel seguente modo per tutti e tre i casi:

```

1 # inizializzo una MLP generica
2 val_p = 0.23
3 hidden_layer_sizes = [16,32,32,16]
4 activation = 'relu'
5 patience = 50
6 max_epochs = 500
7 verbose = True
8 batch_sz = 8
9
10 mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
11                      activation=activation,
12                      early_stopping=True,
13                      n_iter_no_change=patience,
```

```

14     max_iter=max_epochs ,
15     validation_fraction=val_p ,
16     batch_size=batch_sz ,
17     verbose=verbose ,
18     random_state=random_state ,
19     solver='adam' ,
20     learning_rate='adaptive'
21 )

```

Sono stati scelti i seguenti parametri:

- $val\_p = 0.23$ , per avere una percentuale di validation set di circa il 15% del insieme di addestramento. L’insieme di validazione serve per verificare su un piccolo campione le prestazioni del modello.
- Gli strati di neuroni sono composti da  $16 - 32 - 32 - 16$  escludendo il primo che automaticamente copre la dimensione del *training set* e l’ultimo che automaticamente è uno in quanto la predizione può essere o 0 o 1.
- La funzione di attivazione è fissata alla ReLU.
- la *patience* è di 50 cioè il modello si ferma dopo 50 iterazioni in cui la *loss* non migliora di una certa quantità.
- il numero massimo di epoche è di 500 per evitare che il modello spenda troppo tempo ad addestrarsi.
- Il *batch\_size* è stato scelto di 8. Il *batch* è un insieme di punti dell’insieme di addestramento sui quali viene calcolata la *loss* prima dell’aggiornamento dei parametri nell’algoritmo di discesa del gradiente. Generalmente la funzione di perdita viene calcolata su tutto l’insieme di addestramento ma talvolta è preferibile la tecnica dei *mini batch* per favorire l’apprendimento della rete o diminuire il costo computazionale.
- il *solver* scelto è adam, un algoritmo simile alla discesa del gradiente ma ottimizzato.
- il *learning rate* è stato scelto come *adaptive*. Quando la rete migliora le prestazioni allora esso rimane costante, quando non migliora per un po’ esso viene diminuito automaticamente.

Le prestazioni dell’MLP utilizzato sono buone, il modello non fa *overfitting* (vi sono differenze al massimo del 5% tra addestramento e inferenza) e le restanti metriche sono riassunte nelle seguenti tabelle:

Tabella 5.1: MLP per i dati originali

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.93	0.96	0.94	930
1	0.76	0.60	0.66	173
<b>Accuracy</b>			0.90	1103
<b>Macro avg</b>	0.83	0.78	0.80	1103
<b>Weighted avg</b>	0.90	0.90	0.90	1103

Tabella 5.2: MLP per i dati ricampionati con ROS

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.98	0.92	0.95	930
1	0.66	0.88	0.76	173
<b>Accuracy</b>			0.91	1103
<b>Macro avg</b>	0.82	0.90	0.85	1103
<b>Weighted avg</b>	0.93	0.91	0.92	1103

Tabella 5.3: MLP per i dati ricampionati con SMOTE

<b>Label</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Support</b>
0	0.98	0.92	0.95	930
1	0.68	0.89	0.77	173
<b>Accuracy</b>			0.92	1103
<b>Macro avg</b>	0.83	0.91	0.86	1103
<b>Weighted avg</b>	0.93	0.92	0.92	1103

Notiamo che le prestazioni aumentano molto per la classe minoritaria. Pare che l’MLP riesca effettivamente a distinguere le classi 0 e 1, dati gli elevati valori di *precision* e *recall*, molto maggiori di quelli trovati nelle SVM.



# Capitolo 6

## Conclusioni

In generale, il dataset *iranian churn* ha presentato alcuni aspetti abbastanza complicati da gestire. Tra questi, la presenza di molte variabili discrete non ci ha permesso di fare delle ipotesi sulla distribuzione dei dati, e quindi non abbiamo potuto applicare metodi come LDA e QDA. Per quanto riguarda, invece, la riduzione della dimensionalità, si sono riscontrati (dal punto di vista prettamente della visualizzazione) risultati molto migliori con FDA rispetto a quanto visto con PCA. Infatti, nonostante ci fosse una discreta correlazione tra le features, tre componenti principali non si sono rivelate abbastanza per vedere una netta separazione.

La caratteristica del dataset che più di tutte ha influito sulla gestione e sui risultati ottenuti con SVM e MLP è l'elevato sbilanciamento delle classi. Le tecniche di ricampionamento implementate sono riuscite in parte a mitigare questo problema, ma la mancanza di dati reali a disposizione rimane un limite invalicabile.

Le prestazioni delle SVM nel complesso sono buone ma, viste più nel dettaglio, notiamo che riflettono la sproporzione del dataset: esse favoriscono ancora la classe maggioritaria, nonostante i ricampionamenti.

Per quanto riguarda invece la rete neurale implementata, essa riesce a classificare i dati molto meglio. I valori delle metriche relative alla seconda classe si alzano (non sono ancora elevati) e quelli relativi alla prima rimangono stabilmente sopra il 92%.