PROGETTO – CORSO MACHINE LEARNING 2021/2022

UNIVERSITÀ ROMA TRE

Analisi e realizzazione di un sistema di ML che possa classificare la presenza – e in alcuni casi il grado – di retinopatia diabetica nei pazienti a partire da immagini degli occhi.

MARIO CUOMO

Sommario

©MARIO CUOMO

IL PROGETTO

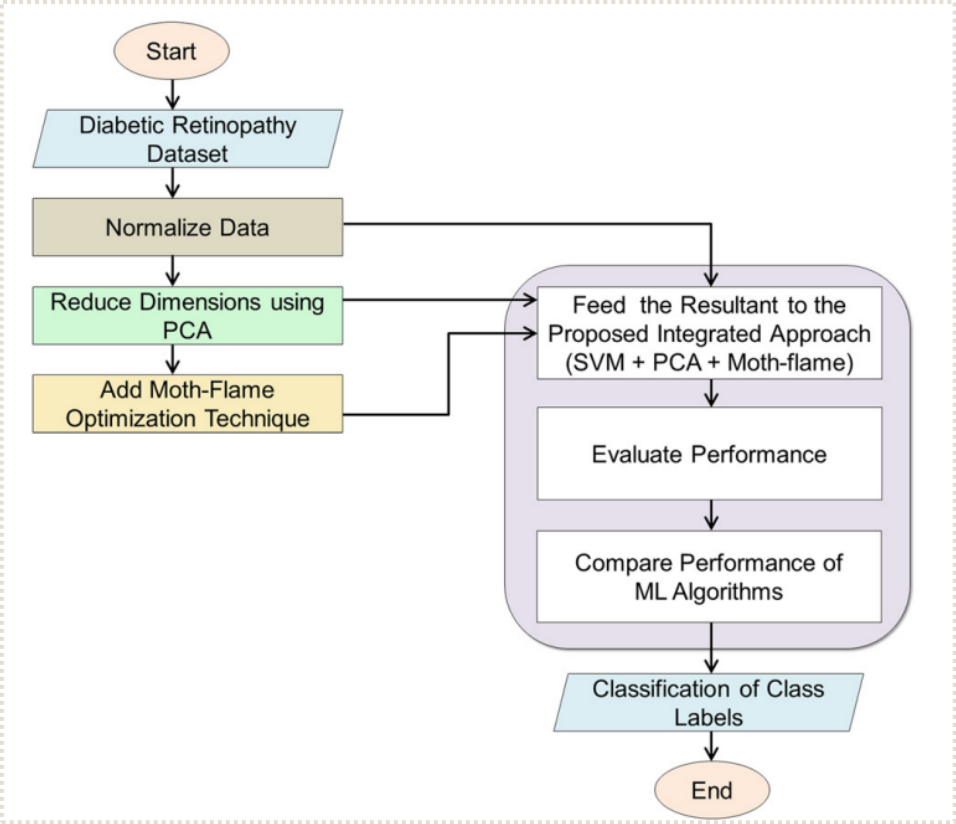
Il progetto è disponibile al seguente indirizzo https://github.com/mariocuomo/progettoML e si pone l'obiettivo di sviluppare un modello di Machine Learning che possa aiutare nella detection automatica della retinopatia diabetica.
La malattia sopra citata è una complicanza del diabete che causa il trattenimento eccessivo dei liquidi nei tessuti – in questo caso quelli oculari – che a lungo termine portano alla generazione di nuovi, ma deboli, vasi sanguigni. La rottura di questi ultimi causa problemi alla vista dei pazienti.
Il problema principale della retinopatia diabetica è che tale malattia si presenta in forma lieve – o addirittura senza alcun sintomo – nei primi stati per poi arrivare a una complicanza con una velocità esponenziale – che in alcuni casi può portare addirittura alla cecità.
Per fortuna la malattia può essere facilmente trattata, se in uno stato lieve asintomatico.
L'idea è quindi di realizzare un modello di Machine Learning che possa risolvere un problema di classificazione.
Sono stati realizzati due studi.

Il primo trae ispirazione da un articolo dal titolo An effective integrated machine learning approach for detecting diabetic retinopathy disponibile a https://www.degruyter.com/document/doi/10.1515/comp-2020-0222.
L'obiettivo dell'articolo è descrivere la realizzazione di un sistema di Machine Learning che possa apprendere – in modo supervisionato – la presenza della retinopatia diabetica a partire da un dataset di features estratte da immagini di occhi. Il dataset è disponibile gratuitamente sull'UCI ML REPOSITORY all'indirizzo https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set.
L'articolo mette a confronto le performance di diversi modelli di classificazione nel caso in cui si applica la normalizzazione dei dati nel dataset, nel caso in cui si applica la riduzione di dimensionalità con la PCA e il caso in cui si effettua una feature selection con l'algoritmo Moth Flame Optimization.
È stato realizzato un notebook Jupyter descrittivo del progetto.
Sono stati utilizzate le tecnologie e metodologie viste a lezione.

Il secondo progetto ha invece una vena più sperimentale.
Invece che utilizzare features estratte dalle immagini, è stata realizzata una Convolutional Neural Network che riceve in input direttamente le immagini e apprende da queste. La rete in questione è la VGG16.

PROGETTO 1

Realizzazione di un modello di Machine Learning che possa identificare la presenza di retinopatia diabetica – classificazione binaria – a partire da feature estratte da immagini di occhi.
Utilizzo di classificatori e metodologie standard.
Di seguito lo schema di analisi.



Flowchart del modello proposto
©screenshot acquisito dall'articolo sopracitato

IL DATASET

Il dataset è composto da 1151 istanze – ognuna delle quali è caratterizzata da 20 features.

Le features in questione sono le seguenti

0	è un valore binario 0 oppure 1 e indica la qualità della foto. 0 la qualità è bassa, 1 la qualità è alta.
1	è un valore binario 0 oppure 1 e indica la presenza di anomalie nella retina. 0 non sono presenti, 1 sono presenti.
$2-7$	sono un valore intero e indicano il numero di micro-aneurismi (MA) che causano la dispersione del sangue nella retina a una determinata confidenza. La feature 2 indica il numero di MA con confidenza 0.2, la feature 3 il numero di MA con confidenza 0.3, … , la feature 7 il numero di MA con confidenza 1.
$8-15$	sono un valore intero e indicano il numero di essudati – liquido derivante dal plasma sanguigno. Segue lo stesso andamento delle feature $2-7$ riguardo il livello di confidenza.
16	è la distanza euclidea tra il centro della macula e il centro del disco ottico.
17	è il diametro del disco ottico.
18	è un valore binario 0 oppure 1 e indica il risultato della classificazione tramite modulazione-ampiezza-modulazione-frequenza. Il valore 1 indica delle immagini retiniche patologiche.
19	è la variabile target per la classificazione binaria. 1 indica che è presente – anche in forma lieve – la retinopatia diabetica.

I CLASSIFICATORI

I classificatori utilizzati sono 4

- DECISION TREE
- NAIVE BAYES
- RANDOM FOREST
- SUPPORT VECTOR MACHINE

DECISION TREE
si costruisce un albero in cui ogni nodo è una feature e ogni arco genitore-figlio rappresenta un possibile valore della feature padre. Si percorre l'albero dalla radice a una foglia in base al valore delle features dell'istanza da classificare.

NAIVE BAYES
si stima la densità di probabilità delle varie classi target e si classifica l'istanza con la classe tale per cui si massimizza la probabilità a posteriori secondo il teorema di Bayes.

RANDOM FOREST
si costruiscono diversi alberi di decisione con approccio bagging – ovvero ogni modello è addestrato in parallelo su un sottoinsieme diverso di dati. Ogni albero effettua la sua predizione e si seleziona la classe che ha ricevuto più voti.

SUPPORT VECTOR MACHINE
piuttosto che calcolare le probabilità delle classi o realizzare un modello esterno, si identifica sul piano la superfice di separazione con il margine più ampio possibile con l'ausilio di support vectors.

METRICHE DI VALUTAZIONE

Per valutare le performance del sistema sono state utilizzate alcune delle metriche principalmente considerate: Precision, Recall, F1-Score, Accuracy e Specificity.

Le istanze che presentano sintomi di retinopatia diabetica sono definite positive, quelle che non la presentano sono definite negative. Se il sistema identifica correttamente una istanza positive si parla di true positive, altrimenti se il sistema sbaglia affermando che una istanza positive è negative si parla di false negative. In maniera speculare si hanno i true negative e i false positive.
Il tutto è riassunto nella tabella seguente

	Realmente positivo	Realmente negativo
Predetto positivo	TP	FP
Predetto negativo	FN	TN

La precision – come suggerisce il nome – descrive quanto il sistema è preciso nell'identificare i true positive. Se il sistema ha una alta precision significa che sbaglia raramente nell'identificare la retinopatia diabetica: quando il sistema afferma che il paziente ha la retinopatia diabetica, è così.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

La recall – in italiano *recupero* – indica il rapporto di istanze positive che sono identificate dal sistema.
Se il sistema ha una alta recall significa che quasi tutte le istanze che presentavano retinopatia diabetica sono state identificate.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

La F1-Score combina tra loro la precision e la recall in una unica metrica.
Questa metrica è la media armonica tra le due.

$$\text{F1} - \text{Score} = \frac{\text{TP}}{\text{TP} + \frac{\text{FN} + \text{FP}}{2}}$$

L'accuracy indica quanto un valore predetto è vicino a quello effettivo: detto in maniera informale è la frazione delle predizioni che sono accurate.

$$\text{Accuracy} = \frac{\#\text{predizioni\_corrette}}{\#\text{predizioni\_totali}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

La specificity misura la proporzione dei true negative ed indica la proporzione delle istanze realmente negative identificate correttamente. Alta specificità significa che il modello sta identificando correttamente la maggior parte delle istanze negative.

$$\text{Specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$

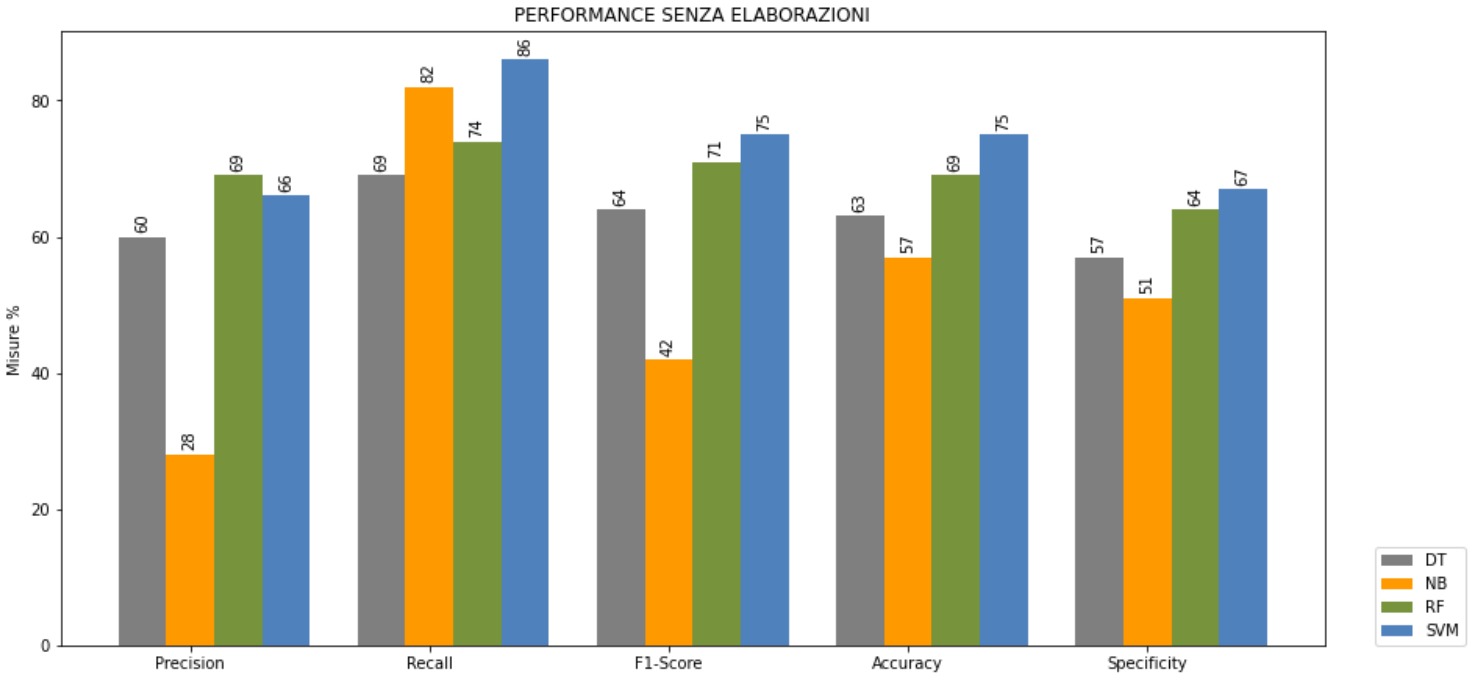
PERFORMANCE CON DATI SENZA ELABORAZIONI

Per prima cosa il dataset è stato diviso in training set e test set in una proporzione dell'80% il primo e 20% il secondo. I classificatori descritti in precedenza sono dichiarati come segue – sfruttando le classi messe a disposizione dalla libreria sklearn per python

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
_y_test=y_test.to_numpy()

decisionTree = DecisionTreeClassifier()
naiveBayes = GaussianNB()
randomForest = RandomForestClassifier(n_estimators=100)
supportVectorMachine = svm.SVC(kernel='linear')
classifiers=[decisionTree, naiveBayes, randomForest, supportVectorMachine]
```

Si calcolano le performance del sistema con le metriche di valutazione precedentemente descritte

```
scores = []

for classificatore in classifiers:
    classificatore.fit(X_train, y_train)
    y_pred = classificatore.predict(X_test)

    tp, fp, fn, tn = recuperaPerformance(_y_test, y_pred)

    score = []

    precision = tp/(fp+tp)
    recall = tp/(fn+tp)
    f1 = (2*precision*recall)/(precision+recall)
    accuracy = (tp+tn)/(tn+fp+fn+tp)
    specifity = tn/(fp+tn)

    score.append(round(precision,2)*100)
    score.append(round(recall,2)*100)
    score.append(round(f1,2)*100)
    score.append(round(accuracy,2)*100)
    score.append(round(specifity,2)*100)

    scores.append(score)
```



Si nota come tutti i classificatori siano caratterizzati da un bassa precision ma una discreta recall. Questo vuol dire che ci sono molti falsi positivi ma pochi falsi negativi ad indicare che il sistema tende a stimare molte presenze di retinopatia diabetica rispetto a quelle realmente presenti.

PERFORMANCE CON DATI NORMALIZZATI

Uno dei problemi da tener conto quando si considerano le varie feature che caratterizza una istanza del dataset è il seguente: se le features sono misurate in scale differenti allora contribuiscono in modo differente al model fitting creando bias intrinseco.

Si applica l'operazione di standardizzazione attraverso lo strumento standardScaler che fa in modo tale che ogni feature – normalizzata indipendentemente dalle altre – abbia media (μ) pari a 0 e varianza (σ) pari a 1.

Considerando un vettore di N elementi, 'operazione di standardizzazione si esegue nel secondo modo

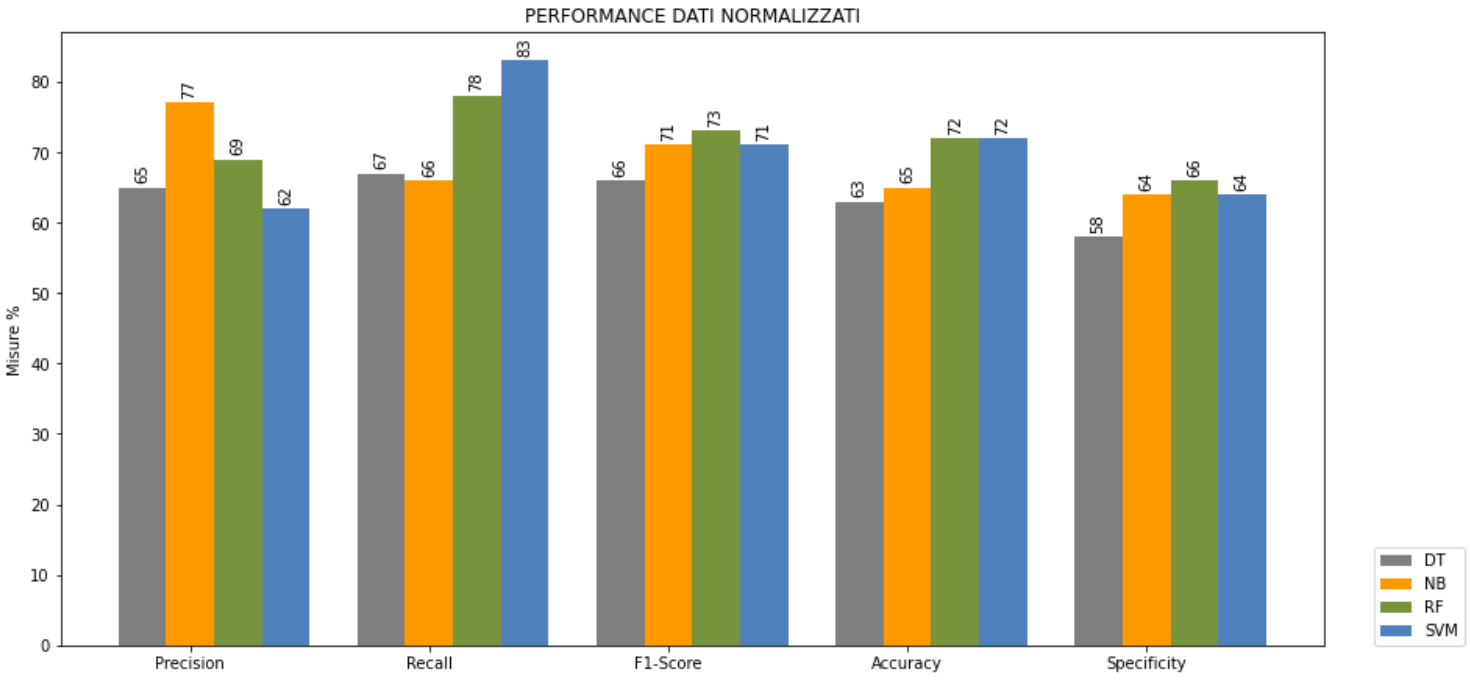
$$x' = \frac{x - \mu}{\sigma}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

È buona norma effettuare un unico addestramento dello standard scaler sul training set ed effettuare l'operazione di transform con lo stesso trained model anche sul test set.

```
standard_scaler = StandardScaler().fit(X_train)

train_x_scaled = standard_scaler.transform(X_train)
test_x_scaled = standard_scaler.transform(X_test)
train_x_scaled = pd.DataFrame(train_x_scaled)
test_x_scaled = pd.DataFrame(test_x_scaled)
```



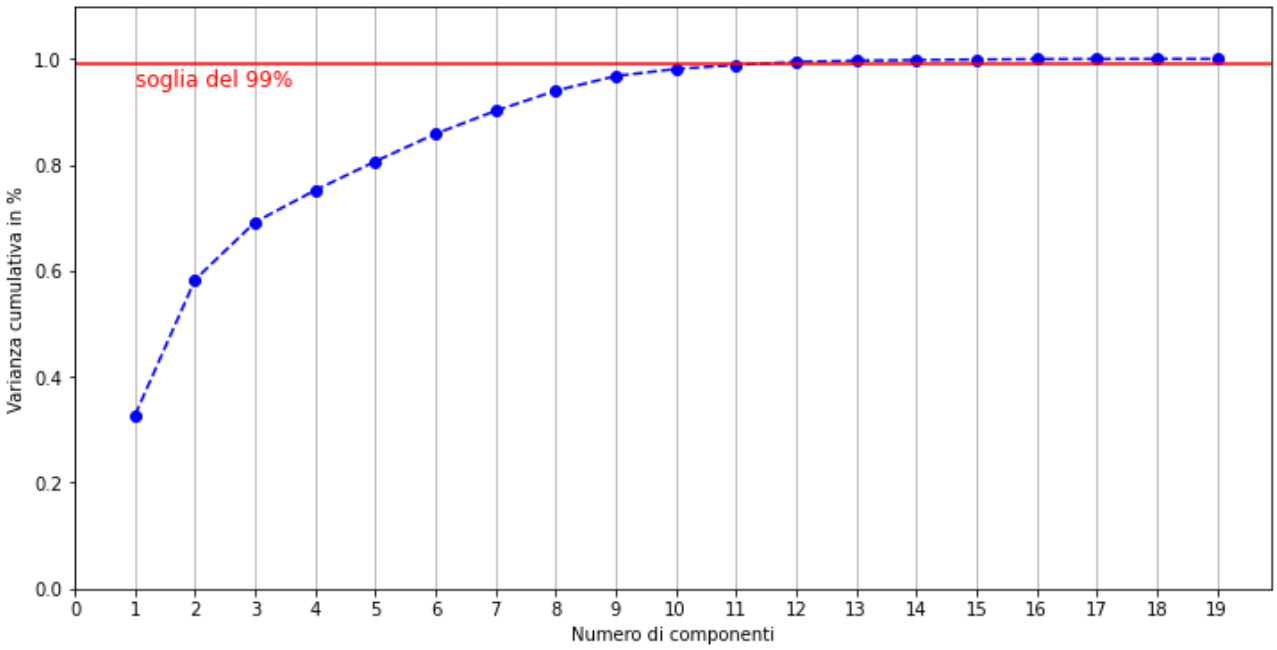
PERFORMANCE CON DATI NORMALIZZATI E APPLICAZIONE PCA

La PCA (Principal Component Analysis) è una tecnica di riduzione della dimensionalità che ha l'obiettivo di ridurre – combinandole tra loro – il numero di feature di un dataset. Inevitabilmente il dataset di output sarà diverso da quello di input ma quello che si vuole fare è mantenere il maggior contenuto informativo possibile. L'idea è quella di sacrificare qualche punto percentuale di accuracy al fine di avere un dataset più semplice, manegevole, facilmente visualizzabile.

L'applicazione della PCA prevede come operazione primaria quella della normalizzazione effettuata al passo precedente; come questa anche nella PCA è necessario effettuare il learning sul training test ed effettuare con lo stesso modello il transform anche sul test set.

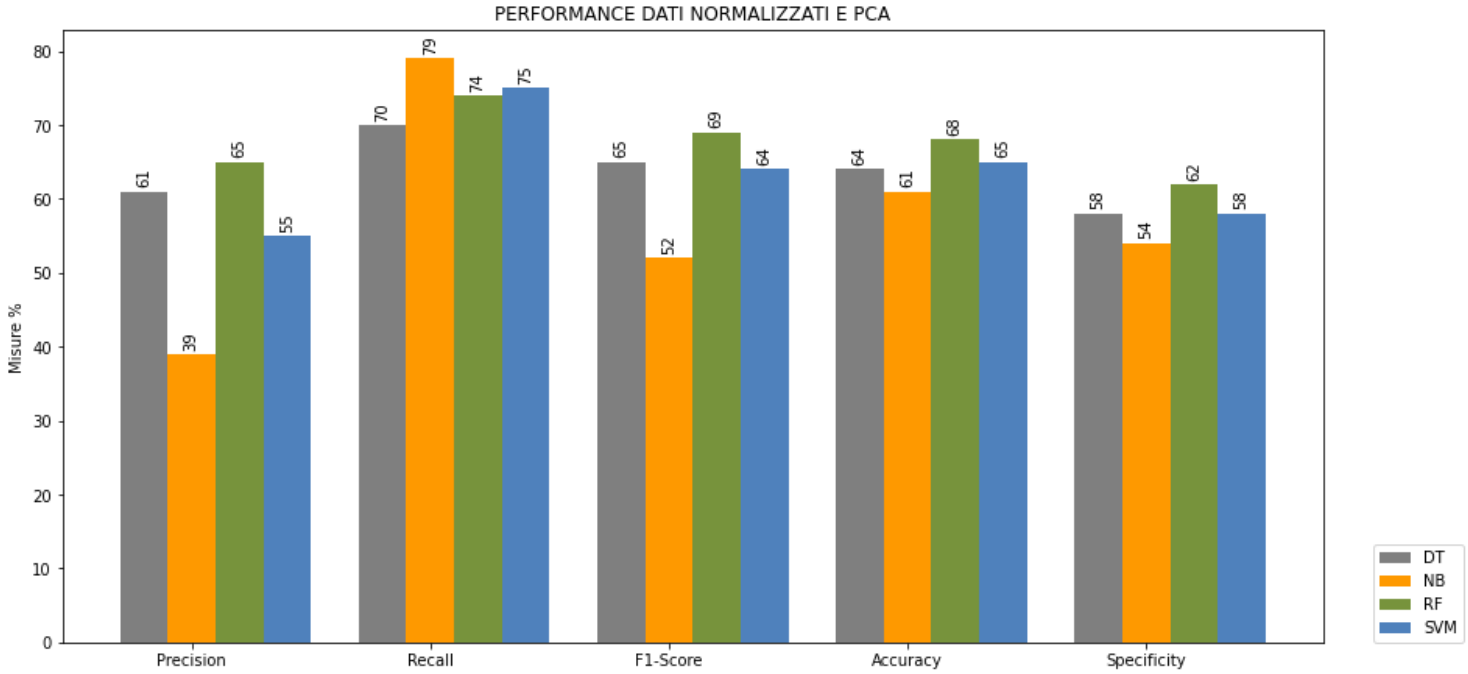
Per mantenere il contenuto informativo più alto possibile – sopra la soglia del 99% - si vede dal grafico seguente che il numero migliore di feature in output è 12 (in realtà 12+target).

```
pca = PCA().fit(train_x_scaled)

fig, ax = plt.subplots()
xi = np.arange(1, 20, step=1)
y = np.cumsum(pca.explained_variance_ratio_)

plt.ylim(0.0,1.1)
plt.plot(xi, y, marker='o', linestyle='--', color='b')

plt.xlabel('Numero di componenti')
plt.xticks(np.arange(0, 20, step=1))
plt.ylabel('Varianza cumulativa in %')
plt.axhline(y=0.99, color='red')
plt.text(1, 0.95, 'soglia del 99%', color = 'red', fontsize=12)
ax.grid(axis='x')
plt.show()
```



```
pca = PCA(n_components = 12)
pca.fit(train_x_scaled)

train_x_scaled_PCA = pd.DataFrame(data = pca.transform(train_x_scaled))
test_x_scaled_PCA = pd.DataFrame(data = pca.transform(test_x_scaled))
```

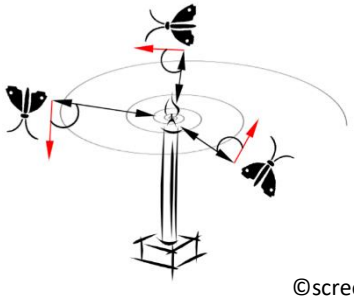
I risultati ottenuti applicando la PCA sono i seguenti



PERFORMANCE CON DATI NORMALIZZATI, APPLICAZIONE PCA E MFO

Il Moth-Flame Optimization (MFO) è un algoritmo metaeuristico per trovare soluzioni sub-ottime a problemi che risultano essere np nella ricerca di soluzioni ottime. Può essere applicato in diversi ambiti ma di seguito verrà utilizzato per effettuare la feature selection. È stato presentato nel 2015 in un articolo dal titolo Moth − flame optimization algorithm: A novel nature − inspired heuristic paradigmdisponibile al seguente indirizzo https://www.sciencedirect.com/science/article/pii/S0950705115002580.

L'algoritmo è una metafora naturale e trae ispirazione dal comportamento delle falene e del loro metodo di orientamento. In sintesi, una falena per effettuare un volo con traiettoria lineare adotta la strategia di mantenere costante l'angolo tra la direzione del moto e la fonte di luce notturna quale la luna. Dato che la luna è a una distanza infinita – considerando le dimensioni e la distanza percorribile di una falena – tale metodo garantisce il percorso lineare. Quello appena descritto è ciò che avviene idealmente. In realtà le falene durante il loro tragitto sono attratte da diverse luci artificiali a breve distanza che ne causano un cambiamento di rotta in una traiettoria a spirale sempre più stretta verso il centro della luce artificiale.



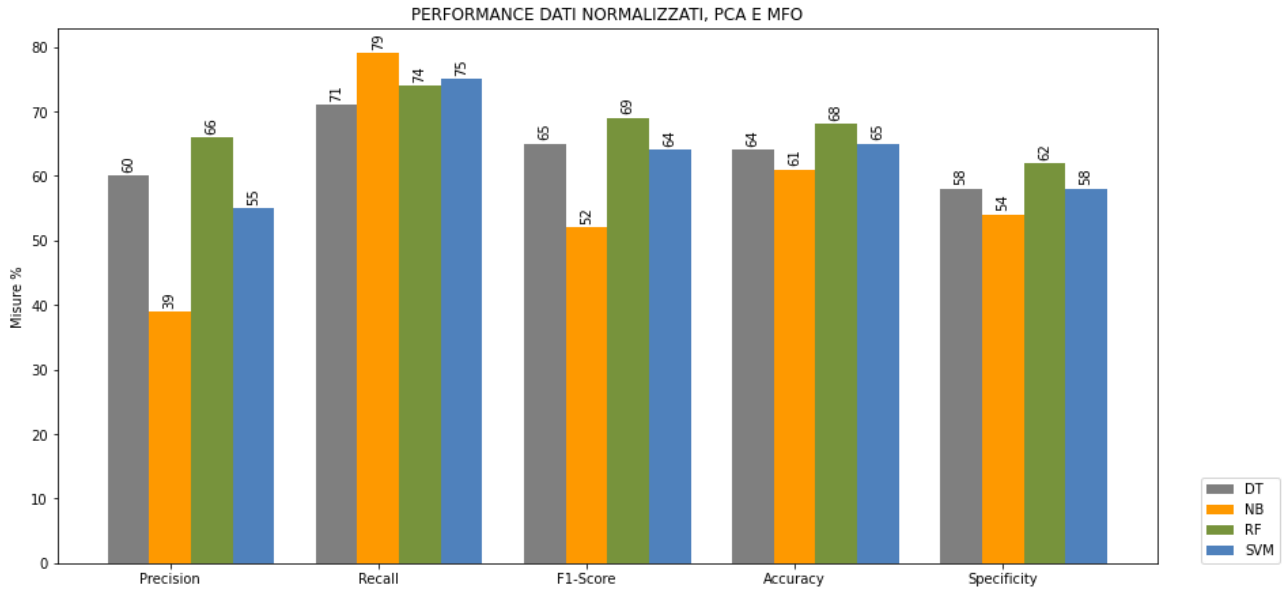
©screenshot acquisito dall'articolo sopracitato

La metafora che si costruisce è la seguente.
Le falene risultano essere le soluzioni candidate che si trovano in uno spazio n-dimensionale se n sono le variabili del problema. Si hanno falene a 12 dimensioni binarie: la posizione $i = 1$ indica che la feature i è selezionata, 0 altrimenti.
Per verificare quanto una posizione della falena – o detto senza metafora, quanto le features selezionate – è appetibile si utilizza una fitness function. La fitness function utilizzata è il valore di accuracy del modello considerando la feature selection espressa dalla falena.

Senza entrare troppo nei dettagli implementativi l'algoritmo può essere descritto come segue

```
PER n iterazioni:
     aggiorna il numero di falene in funzione dell'iterazione i corrente

     calcola il valore di fitness per ogni falena rimasta e ordinale in base
     a tale valore

     aggiorna la posizione delle falene attratte dalle fiamme con una
     funzione coseno
```

Applicando la MTO per la feature selection, si sono ottenuti i seguenti risultati



PROGETTO 2

Questo progetto ha una venatura più sperimentale.
Piuttosto che estrarre delle features dalle immagini e creare un dataset semplificato, si danno in input al sistema direttamente le immagini.
È stata utilizzata una rete neurale convoluzionale, la VGG16.

IL DATASET

Il dataset è composto da 35108 immagini di occhi 17554 pazienti differenti – di ognuno di essi sono infatti presenti le scannerizzazioni di entrambi gli occhi.
La variabile target in questo caso non è binaria ma è un valore intero da 0 a 4 ad indicare il livello di avanzamento dellla retinopatia diabetica nel paziente (0 nulla, 4 massimo rischio).
La distribuzione delle classi è la seguente:

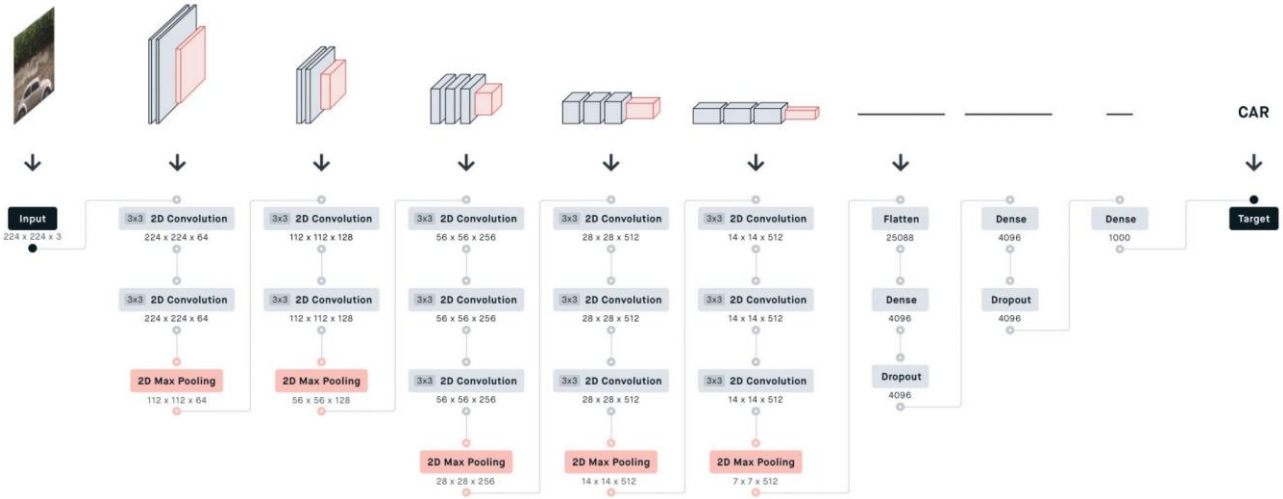
0	25802
1	2438
2	5288
3	872
4	708

Il dateset è scaricabile gratuitamente da un repository Kaggle al seguente indirizzo https://www.kaggle.com/datasets/tanlikesmath/diabetic-retinopathy-resized e per la realizzazione del progetto è stato diviso in training set e test set in una proporzione dell'80% il primo e 20% il secondo.

VGG-16

Il modello di Machine Learning che è stato utilizzato per la classificazione multiclasse è la VGG16 – una rete neurale convoluzionale pensata principalmente per la classificazione/identificazione di immagini in input. Questa rete è stata presentata da Simonyan e Zisserman nel 2014 nell'articolo dal titolo Very Deep Convolutional Networks for Large-Scale Image Recognition.

La rete è composta da 13 strati convoluzionali e 3 completamente connessi.
Essa riceve in input immagini della dimensione $224 \times 224 \times 3$ dove i tre canali rappresentano lo spazio RGB e quindi il processamento anche di immagini a colori.
Di seguito una rappresentazione dell'architettura



Nel primo stack di trasformazioni si incontrano 64 filtri kernel, ovvero filtri convoluzionali bidimensionali della dimensione 3×3 caratterizzati da stride e padding pari a 1. Lo stride indica di quanti passi ci stiamo muovendo in ogni passo nella convoluzione. Dato che la dimensione dell'output è inferiore a quella dell'input, per mantenerla costante si utilizza il padding – un processo di aggiunta simmetrica di zeri alla matrice di input. L'output ha dimensione $224 \times 224 \times 64$ al quale si applica un filtro identico al precedente. Come ultimo step si applica un max pooling layer bidimensionale.
Al termine del primo stack la dimensione delle attivazioni è di $112 \times 112 \times 64$.

Nel secondo stack si utilizzano 128 filtri kernel – sempre utilizzando al termine il max pooling – e di conseguenza la dimensione di uscita è $56 \times 56 \times 128$.

Al terzo stack si utilizzano 256 filtri kernel – sempre bidimensionali – e si applicano 3 volte.
La dimensione delle attivazioni al termine di questo processo è $28 \times 28 \times 256$.

Al quarto stack si utilizzano 512 filtri kernel per tre volte in successione.
Si ottiene un output della dimensione $14 \times 14 \times 512$.

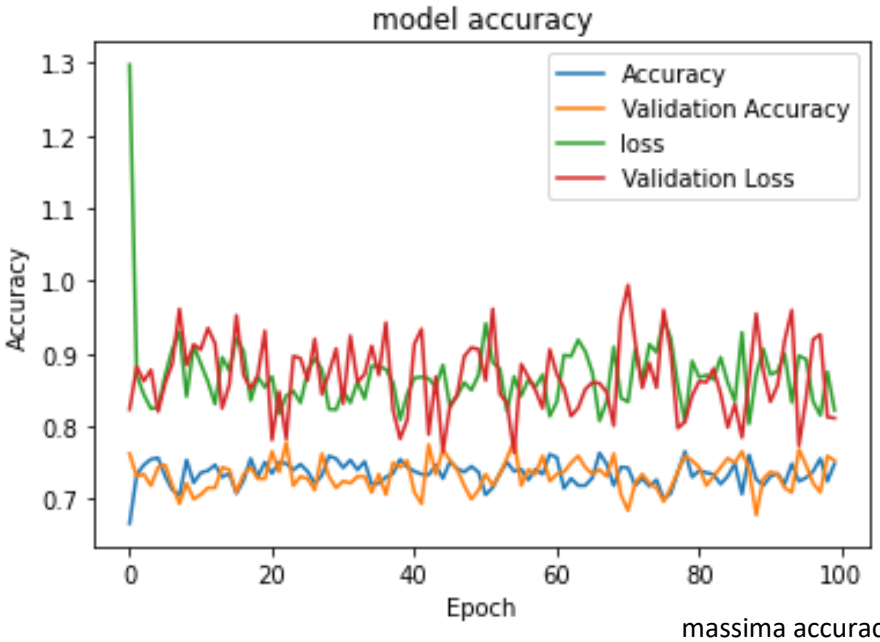
Il quinto stack è una copia del quarto – 3 layer convoluzionali e 1 di max pooling – e produce in output delle attivazioni della dimensione $7 \times 7 \times 512$.

A questo punto seguono 3 strati densi e completamente connessi, preceduti da un flatten layer.
Il flatten layer appiattisce l'output del quinto stack utilizzando 25088 neuroni completamente connessi al primo strato denso di 4096 neuroni. Segue un layer di dropout (opzionale).
Dopo un'altra coppia di strati densi e dropout si ha lo strato di output caratterizzato da x neuroni – con x numero di classi da prevedere.

La rete è stata riprodotta in python con la libreria keras.

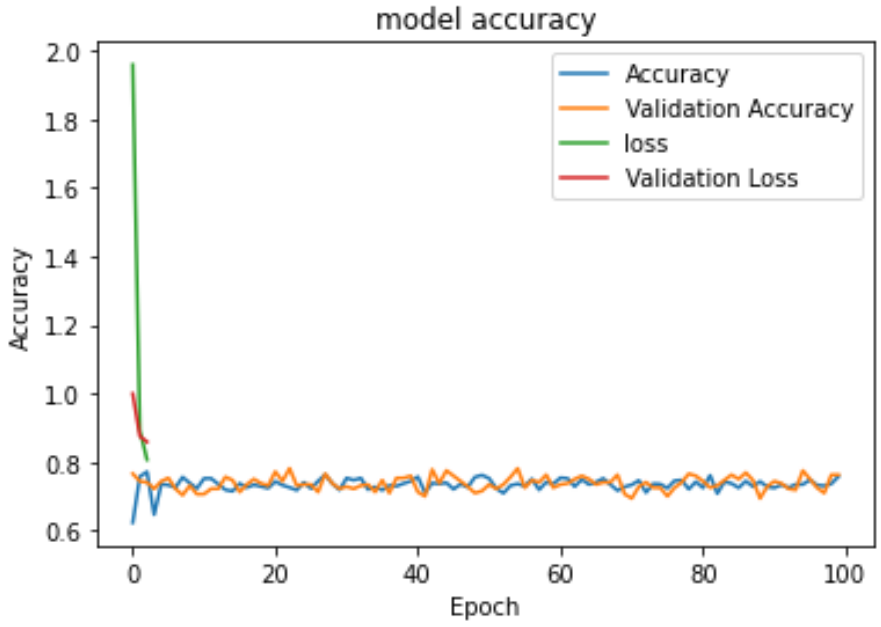
```
model = Sequential()

model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),
          padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
          activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
          activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=5, activation="softmax"))
```

Applicando l'SGD Optimizer – algoritmo di discesa del gradiente con un momentum optimizer per convergere più velocemente – e addestrando il modello con 100 epoche con 32 step per epoca si hanno avuto i seguenti risultati



massima accuracy: 0.7617

Provando ad applicare la Contrast Limited Adaptive Histogram Equalization (CLAHE) come pre-processing alle immagini purtroppo non si hanno avuto forti miglioramenti nell'accuracy finale del sistema.



massima accuracy: 0.7715

CONCLUSIONI

Sommariamente sono soddisfatto del lavoro svolto: più per aver analizzato l'algoritmo di ottimizzazione per la feature selection MFO e aver visto l'architettura della $\mathrm{VGG16}$ che per i risultati ottenuti.
Per quanto riguarda il primo progetto non sono stati raggiunti gli stessi risultati proposti nell'articolo di riferimento – anche cercando di applicare del tuning degli iperparametri.
Per quanto riguarda il secondo progetto sarebbe utile studiare qualche metodo di pre-processing delle immagini in modo tale da essere ottimali per il training della rete neurale $\mathrm{VGG16}$.