Progetto Machine Learning: Analisi Dataset Immagini

Di Emilio Casella matr.204898

L'elaborato sottostante è ottenuto dalla sintesi dei notebook in allegato, per ulteriori dettagli implementativi si rimanda a questi.

Caricamento del Dataset

Il dataset comprende quattro tipologie di immagini da analizzare: jeans,shirts,trousers,watches.
Le immagini vengono caricate in scala di grigio, normalizzate e "flattenizzate" in un array numpy, in modo da agevolare le operazioni successive.Il dataset è formato da (#esempi x (80 x 60), target).

Tuning dei modelli SVC, ADABOOST e KNN

Il Support Vector Machine (SVM) è un algoritmo di apprendimento automatico supervisionato che può essere utilizzato sia per scopi di classificazione che di regressione.
Il Support Vector Machine ha l'obiettivo di identificare l'iperpiano che meglio divide i vettori di supporto in classi. Per farlo esegue i seguenti step:
-Cerca un iperpiano linearmente separabile o un limite di decisione che separa i valori di una classe dall'altro. Se ne esiste più di uno, cerca quello che ha margine più alto con i vettori di supporto, per migliorare l'accuratezza del modello.
-Se tale iperpiano non esiste, SVM utilizza una mappatura non lineare per trasformare i dati di allenamento in una dimensione superiore (se siamo a due dimensioni, valuterà i dati in 3 dimensioni). In questo modo, i dati di due classi possono sempre essere separati da un iperpiano, che sarà scelto per la suddivisione dei dati.

AdaBoost, acronimo di "Adaptive Boosting" rappresenta una tecnica di boosting popolare che aiuta a combinare più "classificatori deboli" in un unico "classificatore forte". Per intenderci, un classificatore debole è semplicemente un classificatore che funziona male, ma funziona meglio di un'ipotesi casuale. Mettendo insieme tanti modelli di questo tipo, l'AdaBoost riesce a generarne uno che complessivamente è migliore dei singoli classificatori deboli presi singolarmente. Come classificatore debole da addestrare l'Adaboost si avvale di tanti alberi decisionali ad un livello di profondità, definiti decision stumps, tanti quanti sono le caratteristiche del modello.Ad ogni iterazione, un nuovo classificatore debole viene introdotto in sequenza e mira a compensare le "carenze" dei modelli precedenti per creare un forte classificatore. L'obiettivo generale di questo esercizio è quello di adattare consecutivamente nuovi modelli per fornire stime più accurate della nostra variabile di risposta.In realtà l'AdaBoost non accetta solo alberi decisionali come weak learners: qualsiasi algoritmo di apprendimento automatico può essere utilizzato come classificatore di base se accetta pesi sul set di allenamento.Si noti che nella configurazione scelta, l'albero ha profondità 1, per emulare i decision stumps; con l'utilizzo dell'algoritmo SAMME.R, che si adatta meglio ad un problema multiclasse.

KNN è un algoritmo di apprendimento supervisionato, il cui scopo è quello di predire una nuova istanza conoscendo i data points che sono separati in diverse classi. Il funzionamento dell'algoritmo K-Nearest Neighbors può essere definito tramite i seguenti step:
1) Si sceglie un valore k con cui prevedere il nuovo data point;
2) Nel caso di grandezze non comparabili si utilizzano tecniche per rendere le misure confrontabili, come la normalizzazione;
3) Si calcola la distanza tra la nuova istanza e i vari data points;
4) Si ordinano le distanze calcolate dalla più piccola alla più grande;
5) Si scelgono le prime K: nel caso si stia svolgendo un problema di regressione, si può restituire con la media delle etichette K. Se si sta svolgendo un problema di classificazione, si sceglierà la classe che include più valori k trovati precedentemente.

AdaBoost è settato con un learning rate piuttosto alto, 1.0 e molti stimatori, 500.
Per quanto concerne SVC, ottengo un C pari a 150 dove C,misura "dimensione" del vettore coefficiente stesso alla funzione, è essenzialmente l'inverso del peso sul termine di "regolarizzazione". Diminuendo C si eviterà il sovradimensionamento forzando i coefficienti a essere piccoli o sparsi, a seconda della penalità. L'aumento eccessivo di C promuoverà il underfitting. Il kernel RBF (Radial Basis Function) è anche chiamato il kernel gaussiano più complesso; contiene un parametro γ: un piccolo valore di γ farà sì che il modello si comporti come un SVM lineare, mentre un grande valore di γ renderà il modello fortemente influenzato dagli esempi dei vettori di supporto.
Gamma,appunto, è un parametro per il kernel RBF. L'aumento della gamma consente un limite decisionale più complesso, qui settato a scale, ovvero 1 / (n_features * variance di X).
KNN utilizzerà 3 vicini e classificherà i punti in base alla distanza di Manhattan.

```
MyClassificatorTrain().transform(X,y)
```

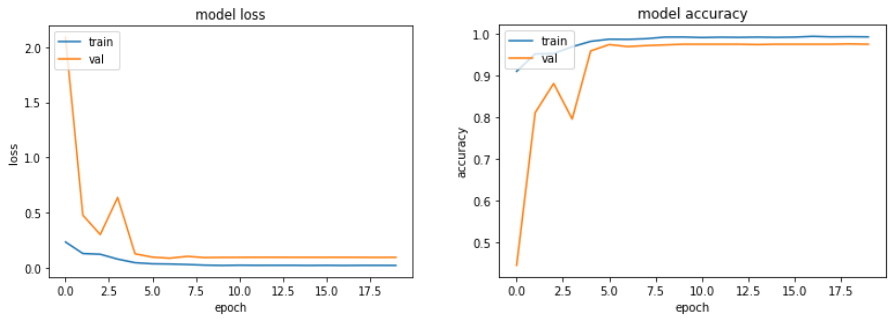
```
Training modelli...
AdaBoost...
Best Param: {'learning_rate': 1.0, 'n_estimators': 500}
Tuning SVC...
Best Param: {'C': 150, 'gamma': 'scale', 'kernel': 'rbf'}
Tuning KNN...
Best Param: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
Fatto!
```

Rete Neurale

Si è scelto di utilizzare una rete convoluzionale, più indicata per l'analisi delle immagini. La rete è composta da tre blocchi sequenziali, composti da layers convoluzionali, di max pooling e normalizzazione batch.
Ciascun livello convoluzionale avrà un passo di uno strides=1, l'impostazione predefinita per Conv2D. I livelli di attivazione sono definiti nel metodo Conv2D e utilizzano un'unità lineare rettificata (ReLU).Tutti i blocchi convoluzionali useranno una dimensione della finestra del filtro di 5x5, tranne il blocco convoluzionale finale, che utilizza una dimensione della finestra di 3x3.Tra il secondo e il terzo livello e dopo quest'ultimo, si è deciso di inserire uno strato di normalizzazione batch al posto dei classici livelli di dropout, per standardizzare e normalizzare i valori ricevuti mitigando l'effetto di gradienti instabili all'interno di reti neurali profonde.Dopo questi blocchi si utilizza la flattenizzazione.Successivamente vengono utilizzati 2 layer densi con dropout di 0.25 alla fine del secondo strato di 100 unità.Infine, poiché si tratta di un problema multi-classe, il nostro livello completamente connesso ha una funzione di attivazione Softmax con 4 neuroni. La funzione di loss scelta è la sparse_categorical_crossentropy, utile per i problemi multiclasse, con ottimizzatore Adam settato ad un learning rate di 0.001.

```
def create_network():
    opt = keras.optimizers.Adam(learning_rate=0.001)
    model_conv = keras.Sequential()
    model_conv.add(layers.Conv2D(25, (5, 5), activation='relu', input_shape=(i, j, 1)))
    model_conv.add(layers.MaxPooling2D((2, 2)))
    model_conv.add(layers.Conv2D(50, (5, 5), activation='relu'))
    model_conv.add(layers.MaxPooling2D((2, 2)))
    model_conv.add(layers.BatchNormalization())
    model_conv.add(layers.Conv2D(70, (3, 3), activation='relu'))
    model_conv.add(layers.MaxPooling2D((2, 2)))
    model_conv.add(layers.BatchNormalization())
    model_conv.add(layers.Flatten())
    model_conv.add(Dense(units=100, activation='relu'))
    model_conv.add(Dense(units=100, activation='relu'))
    model_conv.add(Dropout(0.25))
    model_conv.add(layers.Dense(4, activation='softmax'))
    model_conv.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy']);
    return model_conv
```

Viene studiata la loss su una divisione di prova per 20 epoche, per capire le epoche necessarie al training del modello. Per evitare fenomeni di overfitting, il tutto viene controllato dalla funzione ReduceLROnPlateau che, non appena la rete smette di apprendere, abbassa il learning rate.



Accuratezza sulla 10-fold cross-validation

Come da richiesta progettuale, viene effettuata una validazione incrociata con 10 "sacche". Per assicurare una maggiore affidabilità ai risultati, si utilizza una KFold stratificata, permettendo che una classe di dati non sia sovrappesata soprattutto quando la variabile target è sbilanciata; riducendo,inoltre, bias e varianza.
Per ogni classificatore, verrà salvato il modello che mostra lo score di accuratezza maggiore su una delle dieci divisioni. Si mostra lo score di accuratezza media e la deviazione std ottenuti dal modello sulla 10fold cross validation.
Successivamente, verranno mostrati i risultati ottenuti sul test set relativo al miglior classificatore, tramite lo studio di:
1)Precision: La precisione è l'abilità di un classificatore di non etichettare un'istanza positiva che è in realtà negativa. Per ogni classe è definito come il rapporto tra veri positivi e la somma di veri e falsi positivi;

2)Recall: detta anche sensitivity o true positive, è la capacità di un classificatore di trovare tutte le istanze positive. Per ogni classe è definito come il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi negativi;

3)F-score: è una media armonica ponderata delle metriche Precision e Recall in modo tale che il punteggio migliore sia 1 e il peggiore sia 0;

4)Matrice di confusione: permette di visualizzare meglio il rapporto tra TP,FP,TN,FN;

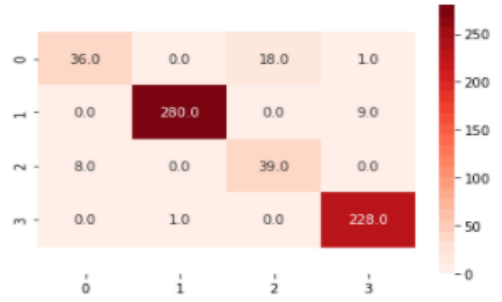
5)Curve di ROC: costruita considerando tutti i possibili valori del test e, per ognuno di questi, si calcola la proporzione di veri positivi (la sensibilità) e la proporzione di falsi positivi. La proporzione di falsi positivi si calcola con la formula standard: 1 - specificità. Congiungendo i punti che mettono in rapporto la proporzione di veri positivi e di falsi positivi (le cosiddette coordinate) si ottiene una curva chiamata curva ROC. L'area sottostante alla curva ROC (AUC, acronimo dei termini inglesi "Area Under the Curve") è una misura di accuratezza diagnostica. L'area sotto la curva può assumere valori compresi tra 0.5 e 1.0. Tanto maggiore è l'area sotto la curva (cioè tanto più la curva si avvicina al vertice del grafico) tanto maggiore è il potere discriminante del test. Per l'interpretazione dei valori dell'area sottostante la curva ROC è possibile riferirsi alla classificazione proposta:

- AUC=0.5 il test non è informativo;
- 0.5<AUC≤0.7 il test è poco accurato;
- 0.7<AUC≤0.9 il test è moderatamente accurato;
- 0.9<AUC<1.0 il test è altamente accurato;
- AUC=1 test perfetto.

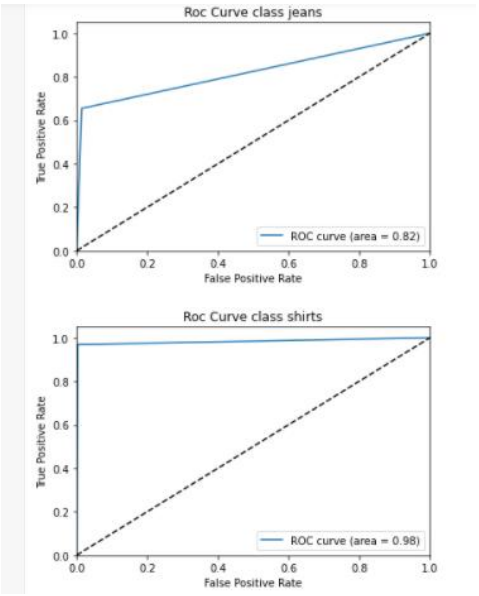
Risultati ottenuti

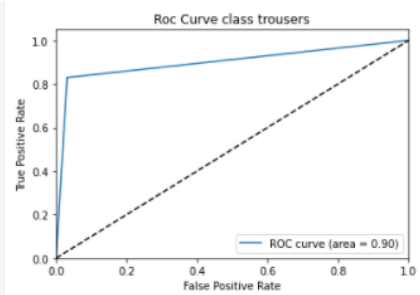
AdaBoost

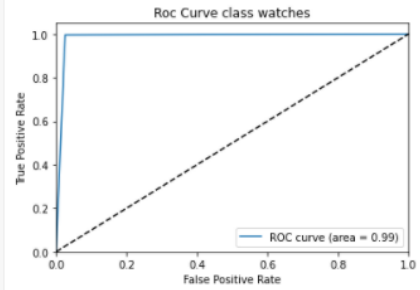
```
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.9178744  0.8921095  0.9178744  0.91465378 0.91935484 0.93064516
 0.94032258 0.92258065 0.91774194 0.92741935]
Ottenendo un'accuratezza media di 0.92 con deviazione standard di 0.012
La migliore accuratezza e' 0.94 ottenuta nello split numero 7
Salvo il modello ottenuto nello split numero 7
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 7
```



	precision	recall	f1-score	support
jeans	0.82	0.65	0.73	55
shirts	1.00	0.97	0.98	289
trousers	0.68	0.83	0.75	47
watches	0.96	1.00	0.98	229
accuracy			0.94	620
macro avg	0.86	0.86	0.86	620
weighted avg	0.94	0.94	0.94	620

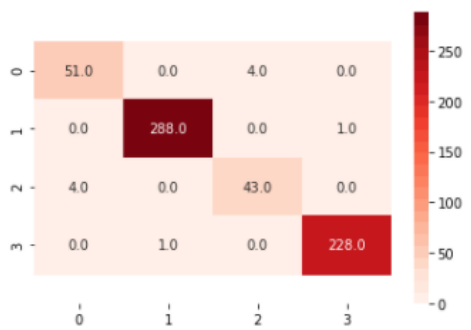




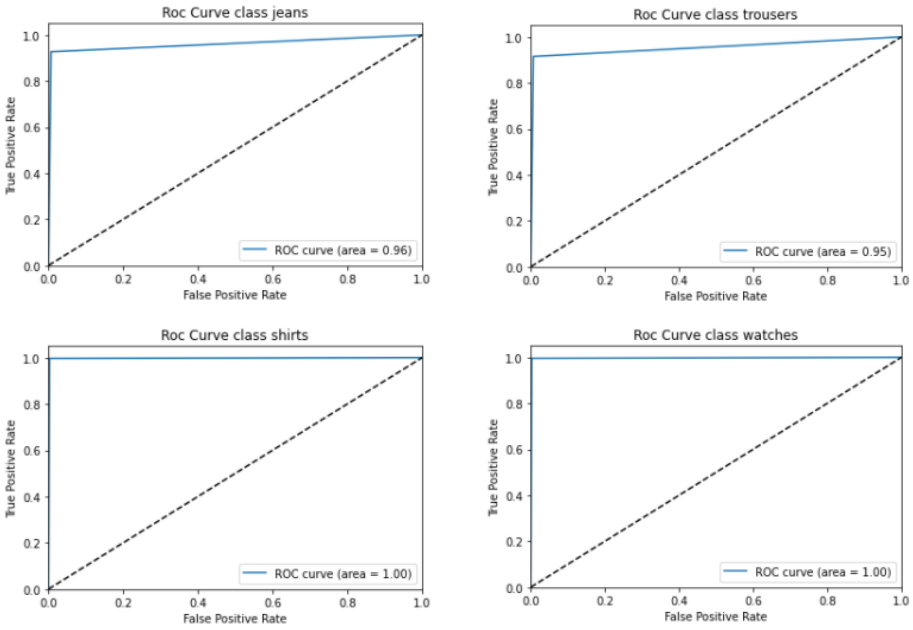


SVC

```
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.97101449 0.97584541 0.97584541 0.97584541 0.98387097 0.98064516
 0.98225806 0.97741935 0.97258065 0.97419355]
Ottenendo un'accuratezza media di 0.977 con deviazione standard di 0.004
La migliore accuratezza e' 0.984 ottenuta nello split numero 5
Salvo il modello ottenuto nello split numero 5
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 5
```

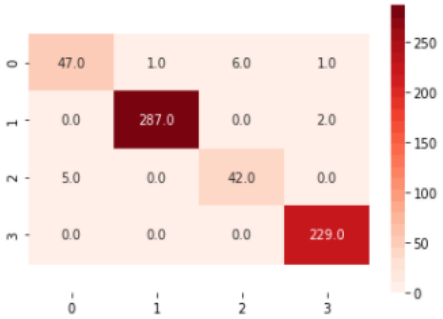


	precision	recall	f1-score	support
jeans	0.93	0.93	0.93	55
shirts	1.00	1.00	1.00	289
trousers	0.91	0.91	0.91	47
watches	1.00	1.00	1.00	229
accuracy			0.98	620
macro avg	0.96	0.96	0.96	620
weighted avg	0.98	0.98	0.98	620

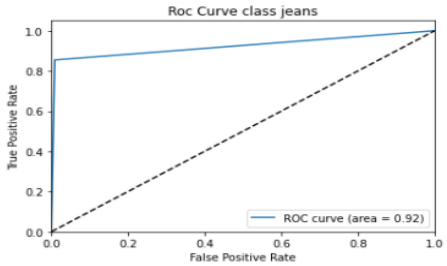


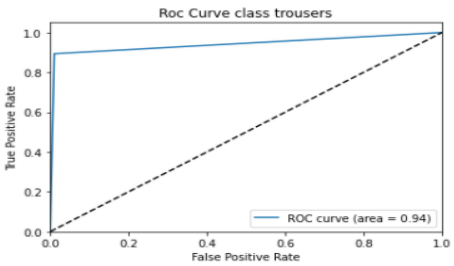
KNN

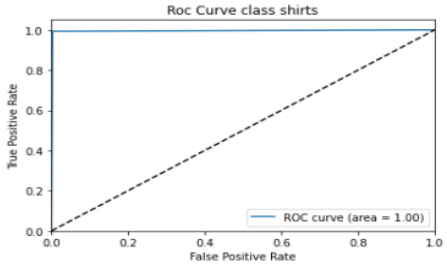
```
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.97101449 0.96940419 0.96618357 0.96940419 0.97580645 0.95483871
 0.97096774 0.96451613 0.94677419 0.96451613]
Ottenendo un'accuratezza media di 0.965 con deviazione standard di 0.008
La migliore accuratezza e' 0.976 ottenuta nello split numero 5
Salvo il modello ottenuto nello split numero 5
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 5
```

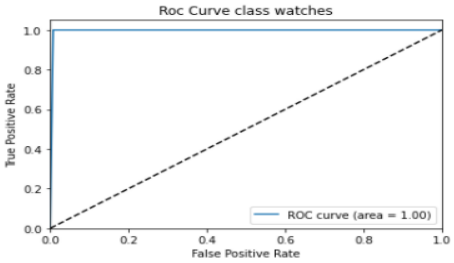


	precision	recall	f1-score	support
jeans	0.90	0.85	0.88	55
shirts	1.00	0.99	0.99	289
trousers	0.88	0.89	0.88	47
watches	0.99	1.00	0.99	229
accuracy			0.98	620
macro avg	0.94	0.94	0.94	620
weighted avg	0.98	0.98	0.98	620



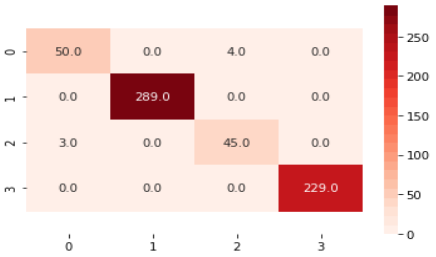




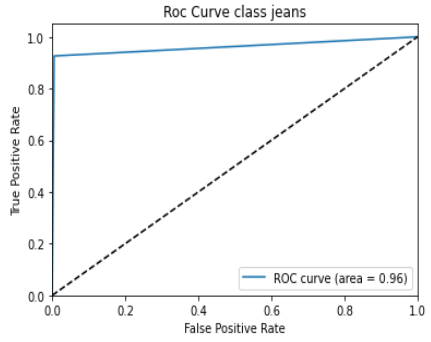


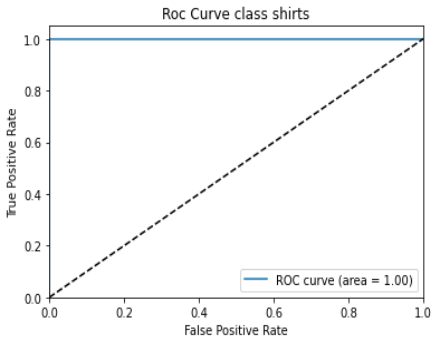
NN

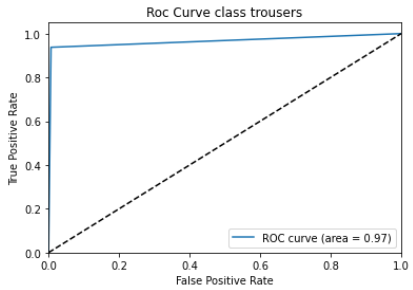
```
Score per fold
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.9758453965187073, 0.9838969111442566, 0.977455735206604, 0.9710144996643066, 0.982258081436157
2, 0.9822580814361572, 0.9854838848114014, 0.9887096881866455, 0.9661290049552917, 0.9741935729980
469]
Ottenendo un'accuratezza media di 0.979 con deviazione standard di 0.007
La migliore accuratezza e' 0.989 ottenuta nello split numero 8
Salvo il modello ottenuto nello split numero 8
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 8
```

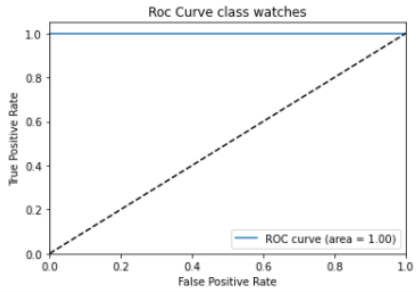


	precision	recall	f1-score	support
0	0.94	0.93	0.93	54
1	1.00	1.00	1.00	289
2	0.92	0.94	0.93	48
3	1.00	1.00	1.00	229
accuracy			0.99	620
macro avg	0.97	0.97	0.97	620
weighted avg	0.99	0.99	0.99	620



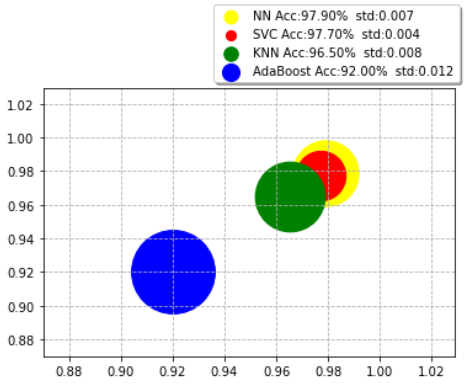






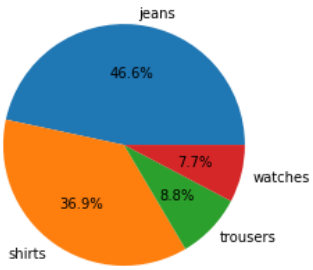
Come si evince da tutti i risultati e i grafici precedenti, il modello migliore per accuratezza è rappresentato dalla rete neurale implementata, seguita da SVC, KNN e AdaBoost.

	accuracy_mean	std
NN	0.979	0.007
SVC	0.977	0.004
KNN	0.965	0.008
AdaBoost	0.920	0.012

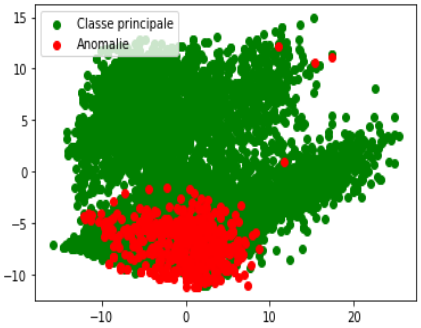


Anomaly Detection

Come richiesto dalla traccia progettuale, si cerca di identificare la classe più numerosa classificandola come corretta; le restanti classi rappresenteranno le anomalie. Si mostra, tramite il grafico sottostante, che la classe jeans è quella che contiene più esempi.



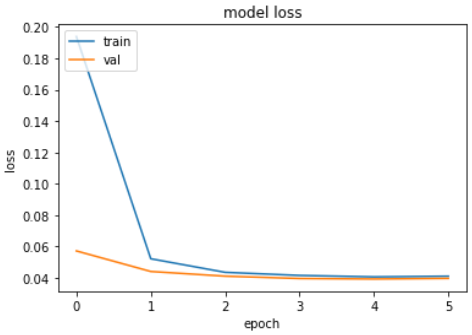
Si mostra ora la composizione del nuovo dataset ottenuto, tramite il grafico sottostante ottenuto tramite PCA. L'analisi delle componenti principali (PCA) è un metodo che rientra nei problemi di trasformazione lineare che viene ampiamente utilizzata in diversi campi, soprattutto per l'estrazione delle caratteristiche e la riduzione della dimensionalità. PCA permette di trovare le direzioni della massima varianza nei dati ad alta dimensione e di proiettarle su un nuovo sottospazio con dimensioni uguali o inferiori a quello originale. Utilizzando la proiezione matematica, il set di dati originale, che potrebbe aver coinvolto molte variabili, viene ad essere interpretato solo da poche variabili (dette componenti principali). L'output di PCA sono proprio queste componenti principali, il cui numero è inferiore o uguale al numero di variabili originali.

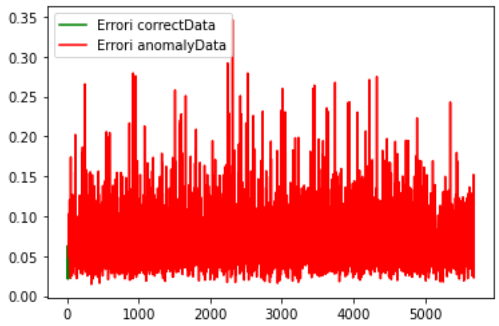


Si effettuano due tipologie di analisi:
1) Approccio tramite rete densa con il compito di comprimere e riespandere l'output. Il modello viene allenato per poche epoche,6, e presenta una loss mse, che si comporta molto bene nella "ricostruzione" ed analisi delle immagini;
2) Approccio tramite un classificatore PCA.

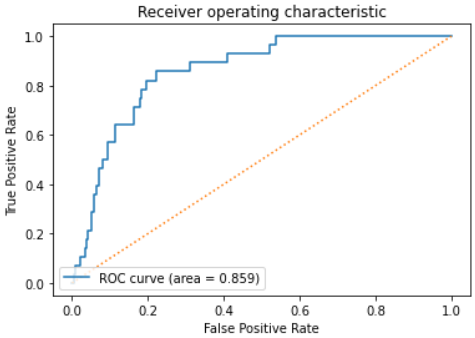
```
my_init = keras.initializers.glorot_uniform(seed=1)
autoenc = keras.models.Sequential()
autoenc.add(keras.layers.Dense(input_dim=i*j, units=100, activation='tanh', kernel_initializer=my_init))
autoenc.add(keras.layers.Dense(units=2400,activation='tanh', kernel_initializer=my_init))
autoenc.add(keras.layers.Dense(units=100,activation='tanh', kernel_initializer=my_init))
autoenc.add(keras.layers.Dense(units=50,activation='tanh', kernel_initializer=my_init))
autoenc.add(keras.layers.Dense(units=100,activation='tanh', kernel_initializer=my_init))
autoenc.add(keras.layers.Dense(units=2400,activation='tanh', kernel_initializer=my_init))
autoenc.add(keras.layers.Dense(units=i*j,activation='tanh', kernel_initializer=my_init))
autoenc.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,patience=1, min_lr=0.000001)
```

La divisione, per entrambi i classificatori, è caratterizzata da un set che presenta molte anomalie e minori esempi corretti, in modo tale da avere risultati affidabili sia in fase di train che di test di entrambi gli approcci. Il grafico sottostante permette di studiare la soglia degli errori commessi su entrambe le classi e definirne un limite, soprattutto per la classe corretta. Si può notare una soglia massima di errore sulla classe corretta dello 0,07 circa; un discreto risultato.

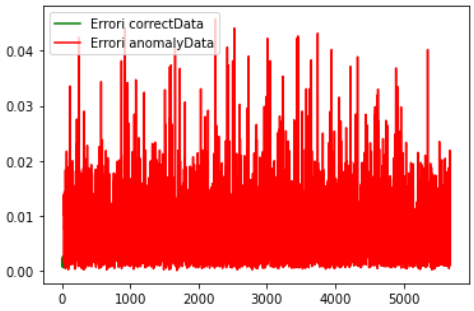


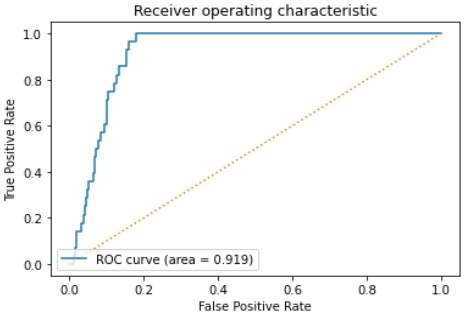


Le considerazioni precedenti, sono confermate dalla curva di roc sottostante, presentando un risultato moderatamente accurato.



Si passa ora all'analisi tramite PCA. Il tuning della rete viene effettuato testando un insieme di numeri diversi di features, portando alla scelta di 496.
La soglia dell'errore sulla classe positiva è molto bassa, quasi vicina allo zero, mostrando un risultato migliore della metodologia precedente. La curva di ROC conferma tutto questo, mostrando un risultato altamente accurato.





Viene salvato il modello PCA prodotto, il migliore in questo contesto.
Impostando una soglia di errore massima, si può notare come l'accuratezza del modello salvato sia quasi perfetta, confermando ulteriormente le considerazioni precedenti.

```
y = dataFrame["class"]
X = dataFrame.drop("class",axis=1).values
anomaly_detection=loadM("BestAnomayModel")
pred=anomaly_detection.inverse_transform(anomaly_detection.transform(X))
err= mean_squared_err(X,pred)
y_pred=(np.array(thresholdErr(err,0.0001)))
accuracy_anomaly=accuracy_score(y,y_pred)*100
print(str(np.round(accuracy_anomaly,3))+" % ")
```

```
99.468 %
```

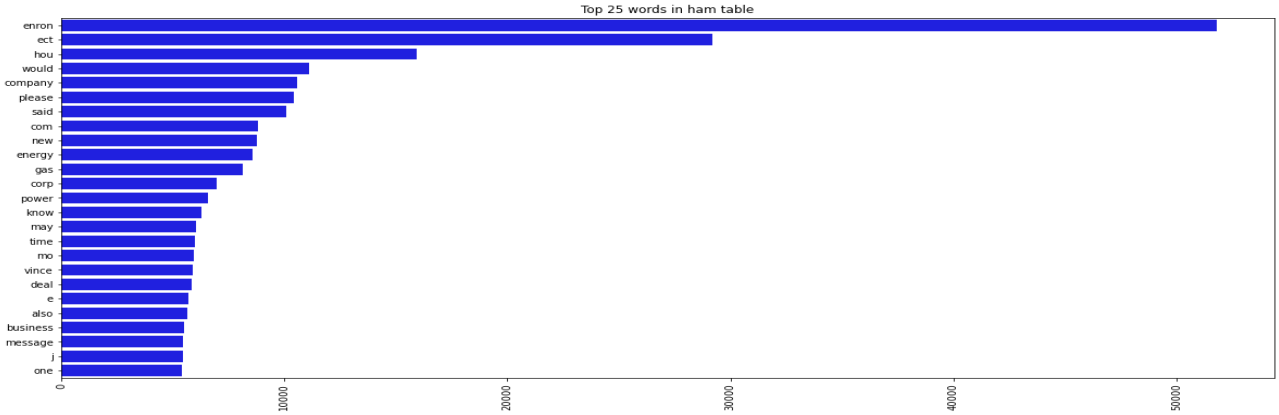
Progetto Machine Learning: Analisi Dataset Testuale

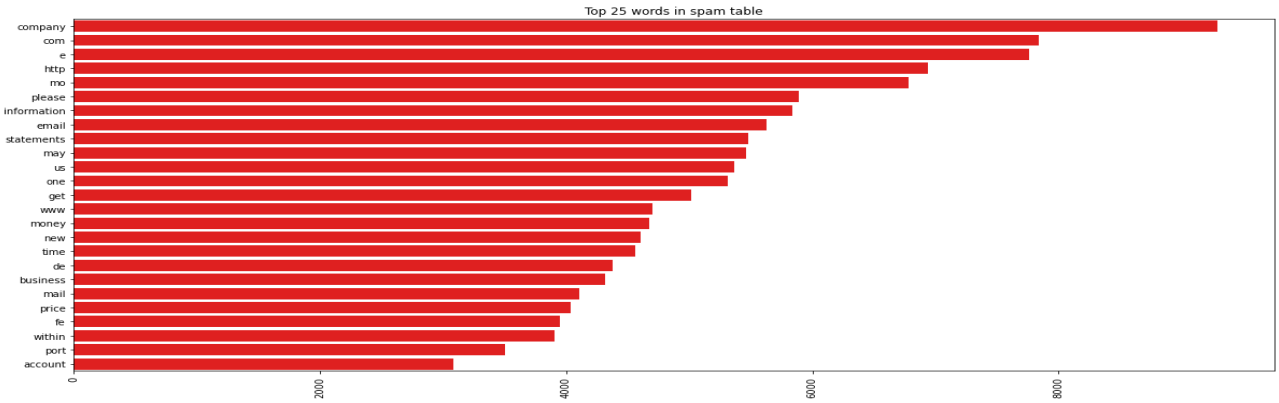
Il dataset è composto da mail ham e spam che, per mia scelta progettuale, vengono opportunamente caricate in un dataframe pandas.
Le operazioni successive permettono di "ripulire" il testo da tutto ciò che viene considerato superfluo, al fine di consentire una migliore analisi.

	text	target
0	Subject: fw : new generation report for june 2...	0
1	Subject: confirmation of meeting\nvince : than...	0
2	Subject: re : associate prc - input needed\ni ...	0
3	Subject: re : gwen koepke\nanne ,\nthanks for ...	0
4	Subject: become a stud in bed\nfact :\nin a po...	1

	text	target
0	[fw, new, generation, port, june, guys, intest...	0
1	[confirmation, meetingvince, thanks, introduci...	0
2	[associate, prc, input, neededi, houston, th, ...	0
3	[gwen, koepkeanne, thanks, contacting, matter,...	0
4	[become, stud, bedfact, poll, conducted, congl...	1

Vengono ora mostrate le top 25 parole presenti per entrambe le tipologie di mail.





Si crea un embeddings per la tabella nel seguente modo:
1) Creo un dizionario contenente al max le 200 parole più frequenti per entrambe le tipologie di mail;
2) Creo un vettore che, per ogni file, analizza le parole e la converte con l'indice che la rappresenta nel dizionario, se questa è presente, altrimenti col valore zero;
Otterrò, per il dataset in esame, una rappresentazione vettoriale di grandezza 305.

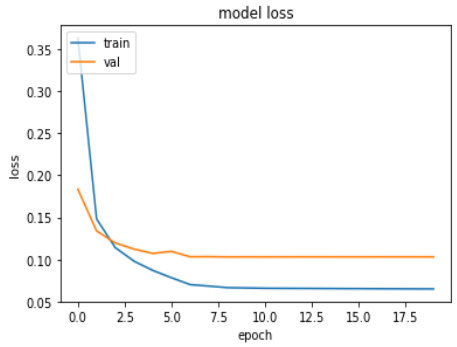
	text	target
0	[1, 0, 0, 1, 0, 1, 0, 1, 6, 0, 0, 0, 3, 0, 0, ...	0
1	[0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 1, 0, 1, 0, 1, ...	0
2	[18, 10, 4, 2, 0, 3, 0, 0, 0, 0, 0, 2, 0, 0, 0...	0
3	[2, 1, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	0
4	[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, ...	1

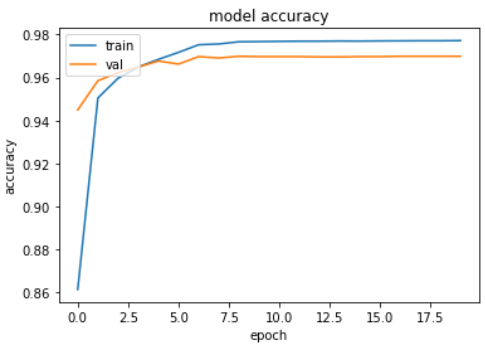
Rete Neurale

La rete neurale, che verrà utilizzata, sfrutta vari livelli densi, particolarmente indicati in questa casistica. Si utilizza una loss 'binary_crossentropy', mentre l'ultimo layer utilizza una funzione di attivazione sigmoidale, ottima nella classificazione binaria.

```
def create_network():
    opt = keras.optimizers.Adam(learning_rate=0.0001)
    model = Sequential()
    model.add(keras.layers.Dense(128, activation='relu',input_dim=305))
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dense(32, activation='relu'))
    model.add(layers.Dense(10, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer=opt,loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Viene studiata la loss su una divisione di prova per 20 epoche, per capire il numero necessario al training del modello. Per evitare fenomeni di overfitting, il tutto viene controllato dalla funzione ReduceLROnPlateau che, non appena la rete smette di apprendere, abbassa il learning rate.





La scelta ricade su 20 epoche.

Tuning dei modelli SVC,ADABOOST e KNN

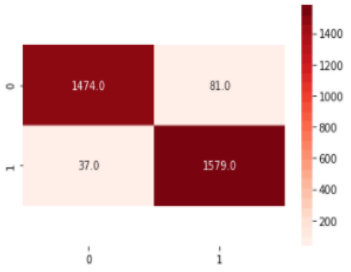
Le restanti tipologie di classificatori, con annessi passaggi, sono i medesimi affrontati per l'analisi del dataset precedente, non necessitando di ulteriori spiegazioni.

```
Training modelli...
AdaBoost...
Best Param: {'learning_rate': 1.0, 'n_estimators': 500}
Tuning SVC...
Best Param: {'C': 200, 'gamma': 'scale', 'kernel': 'rbf'}
Tuning KNN...
Best Param: {'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}
Fatto!
```

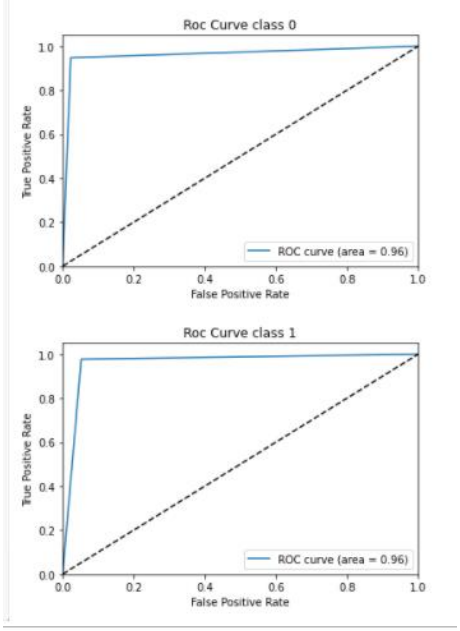
Accuratezza sulla 10-fold cross-validation

AdaBoost

```
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.96278776 0.95868811 0.95425868 0.9615142  0.96088328 0.95425868
 0.96277603 0.96119874 0.95615142 0.95741325]
Ottenendo un'accuratezza media di 0.959 con deviazione standard di 0.003
La migliore accuratezza e' 0.963 ottenuta nello split numero 1
Salvo il modello ottenuto nello split numero 1
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 1
```

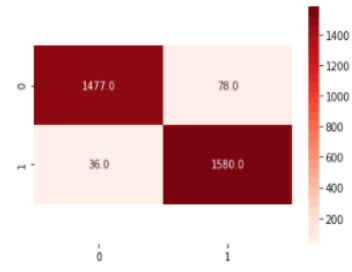


	precision	recall	f1-score	support
0	0.98	0.95	0.96	1555
1	0.95	0.98	0.96	1616
accuracy			0.96	3171
macro avg	0.96	0.96	0.96	3171
weighted avg	0.96	0.96	0.96	3171

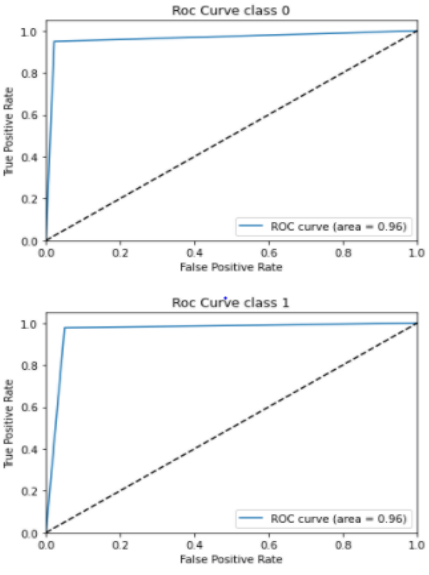


SVC

```
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.9640492  0.95931883 0.95457413 0.96214511 0.96214511 0.9533123
 0.96246057 0.96119874 0.95835962 0.95930599]
Ottenendo un'accuratezza media di 0.96 con deviazione standard di 0.003
La migliore accuratezza e' 0.964 ottenuta nello split numero 1
Salvo il modello ottenuto nello split numero 1
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 1
```

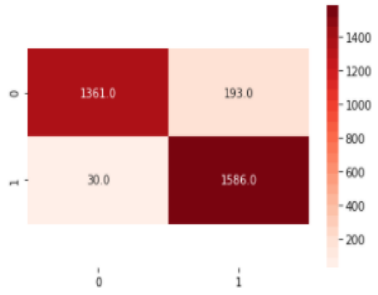


	precision	recall	f1-score	support
0	0.98	0.95	0.96	1555
1	0.95	0.98	0.97	1616
accuracy			0.96	3171
macro avg	0.96	0.96	0.96	3171
weighted avg	0.96	0.96	0.96	3171

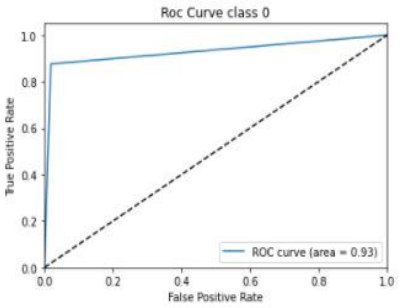


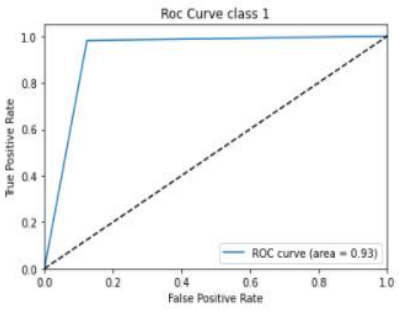
KNN

```
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.92210659 0.92336802 0.9148265  0.92681388 0.92807571 0.91514196
 0.92082019 0.92176656 0.929653   0.92586751]
Ottenendo un'accuratezza media di 0.923 con deviazione standard di 0.005
La migliore accuratezza e' 0.93 ottenuta nello split numero 9
Salvo il modello ottenuto nello split numero 9
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 9
```



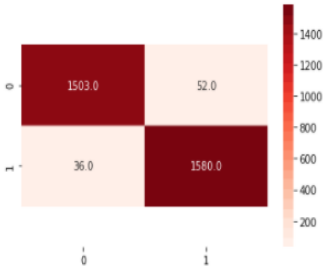
	precision	recall	f1-score	support
0	0.98	0.88	0.92	1554
1	0.89	0.98	0.93	1616
accuracy			0.93	3170
macro avg	0.93	0.93	0.93	3170
weighted avg	0.93	0.93	0.93	3170



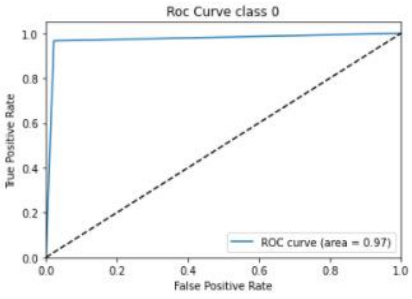


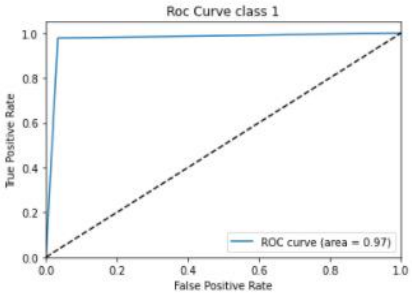
NN

```
Score per fold
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.9722484946250916, 0.968464195728302, 0.960567831993103, 0.970346987247467, 0.9646687507629395, 0.9
596214294433594, 0.9706624746322632, 0.9662460684776306, 0.9630914926528931, 0.9649842381477356]
Ottenendo un'accuratezza media di 0.966 con deviazione standard di 0.004
La migliore accuratezza e' 0.972 ottenuta nello split numero 1
Salvo il modello ottenuto nello split numero 1
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 1
```



	precision	recall	f1-score	support
0	0.98	0.97	0.97	1555
1	0.97	0.98	0.97	1616
accuracy			0.97	3171
macro avg	0.97	0.97	0.97	3171
weighted avg	0.97	0.97	0.97	3171





Da tutti i risultati e grafici precedenti, si evince come la rete neurale rappresenti il miglior modello.

Embeddings tramite Doc2Vec

Vediamo ora, come si comporta il miglior modello ottenuto, utilizzando un embeddings ricavato da una rete neurale tipicamente utilizzata per l'analisi testuale: Doc2Vec.
Prima di inoltrarsi nella spiegazione di come effettivamente questo approccio sia stato utilizzato per gli scopi da noi prefissati, è necessario fornire un breve background teorico. Doc2Vec in realtà è un metodo derivante dal più noto Word2Vec che consiste in un gruppo di modelli che sono usati per effettuare del word embedding, ovvero una traduzione di parole o frasi in vettori composti da numeri reali. Questi modelli non sono altro che delle reti neurali a due livelli che sono addestrate (tramite un approccio non supervisionato) per ricostruire i contesti linguistici delle parole. Word2Vec prende come input un grande frammento di testo e costruisce uno spazio vettoriale assegnata ad un corrispondente vettore nello spazio, seguendo un certo criterio. Infatti i vettori (rappresentanti le parole) vengono posizionati nello spazio

cosicché le parole che risultino "simili" all'interno del frammento di testo, siano collocate vicine tra loro nello spazio stesso. L'idea infatti è quella di riuscire ad incapsulare relazioni differenti tra le parole, comead esempio sinonimi, contrari o analogie.Il vero obiettivo sarà quello di imparare i pesi del livello di hidden, così da usarli come codifica di ogni singola parola (word vector). Nello specifico, esistono due metodologie per poter creare il modello che permette la codifica delle parole in vettori:

• Skip-Gram model: in cui si prendono coppie di parole dal testo e si addestra una rete neurale con un livello di nodi di hidden sulla base del finto task in cui, a partire da una parola in input, la rete restituisce la distribuzione di probabilità delle parole vicine (all'interno del testo) all'input. In altre parole, la rete restituirà alte probabilità per le parole che tipicamente compaiono "vicine" alla parola in input. Essendo un finto task però, ad essere realmente importanti saranno i pesi tra i nodi di input e i nodi di hidden che saranno usati come word embedding. Quindi se lo strato di hidden possiede 300 neuroni, la rete restituirà un vettore di dimensione pari a 300, per ciscuna parola;

• A continuous bag of words (CBOW): anch'essa fa uso di una rete neurale con un livello di nodi di hidden. Il finto task in questo caso è basato sul predire una parola centrale, a partire da un insieme di parole di input (appartenenti allo stesso contesto). Anche qui, saranno poi i pesi tra il livello di input e di hidden ad essere usati come word embeddings per i termini dati in ingresso alla rete.

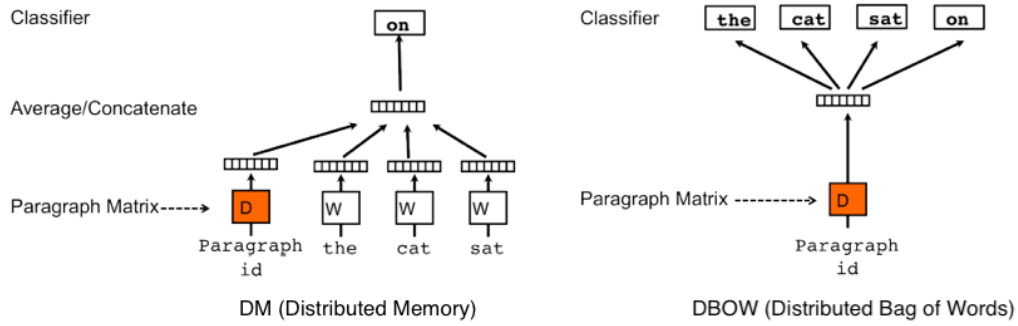
L'obiettivo del Doc2Vec è di creare una rappresentazione numerica di un intero documento (o paragrafo, nel nostro caso di un tweet) a prescindere dalla sua lunghezza. Il principio usato è semplice ed intelligente: si fa uso del modello word2vec e si aggiunge un altro vettore, detto Paragraph ID. Quindi dopo aver addestrato la rete neurale, si a vranno oltre che i word vectors (la rappresentazione vettoriale delle parole) anche un document vector (la rappresentazione vettoriale del documento).

Anche nel caso del Doc2Vec esistono due metodologie analoghe rispettivamente al CBOW e lo Skip-Grim:

• Distributed Memory version of Paragraph Vector (PV-DM): che agisce come una memoria che ricorda cosa manca esattamente all'interno del contesto in questione o come l'argomento del paragrafo. Mentre i word vectors rappresentano il concetto di una parola, il document vector intende rappresentare il concetto di un documento.

• Distributed Bag of Words version of Paragraph Vector (PV-DBOW): che risulta essere più veloce e che consuma meno memoria (contrariamente a quanto accade allo skip-gram nel word2vec) in quanto non c'è alcun bisogno di salvarsi i word vectors (rappresentazioni vettoriali delle parole). Dopo l'addestramento infatti, basta fornire il paragraph ID (detto anche tag) per ricevere il document vector.

Si utilizzerà l'algoritmo PV-DBOW, con un vettore di taglia 500 per rappresentare il documento. La predizione avrà una finestra di 2 parole e si utilizzeranno 10 epoche di train.



DM (Distributed Memory) DBOW (Distributed Bag of Words)

```
]: df_doc2vec=pd.read_pickle("/home/emiliocasella/Scrivania/proj204898/txt/cleanwords")
   words=df_doc2vec.text.to_list()
   target=df_doc2vec.text.to_numpy()
   documents = [TaggedDocument(words[i], target[i]) for i in range(len(target))]
   model = gensim.models.Doc2Vec(documents, vector_size=500, window=2,epochs=10, min_count=1, workers=core
   model.save('d2vec'+'.model')
   print("Fatto!")

   Fatto!
```

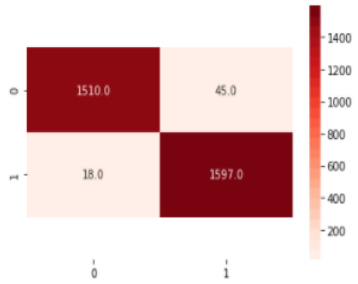
In base al modello di embeddings creato, si ottiene la seguente conversione per ogni documento.

```
MyDoc2vec().transform(df_doc2vec,['text'],"d2vec.model")
df_doc2vec.head()
```

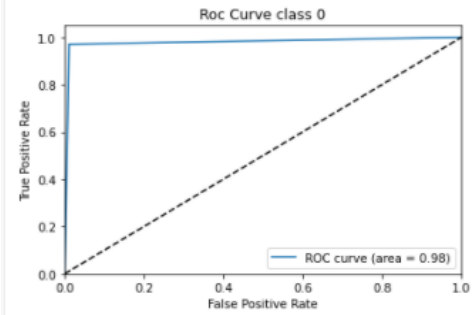
	text	target
0	[0.17600273, -0.33752394, 0.10832454, -0.03667...	0
1	[-0.2561136, 0.08447421, 0.090536326, -0.05035...	0
2	[-0.8033857, 0.3208517, 0.7965166, -0.14490508...	0
3	[-0.3176085, 0.45610327, 0.20239797, 0.3979420...	0
4	[-0.74756914, 0.009872089, -0.020475907, 1.624...	1

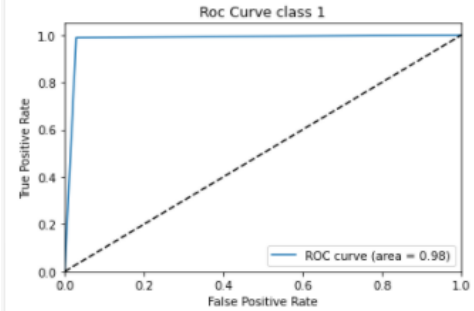
La rete neurale utilizzata come classificatore è simile alla precedente, con l'aggiunta di un primo livello a 256, avendo a che fare con un vettore di dimensione maggiore.

```
Score per fold
L'accuratezza per ogni sacca della cross validation è rappresentata dal seguente vettore
[0.9757174253463745, 0.9728792309761047, 0.960567831993103, 0.9801262021064758, 0.9675078988075256,
0.969085156917572, 0.9637224078178406, 0.967823326587677, 0.9706624746322632, 0.9709779024124146]
Ottenendo un'accuratezza media di 0.97 con deviazione standard di 0.005
La migliore accuratezza e' 0.98 ottenuta nello split numero 4
Salvo il modello ottenuto nello split numero 4
Si mostra ora la matrice di adiacenza ottenuta sul test set dello split numero 4
```



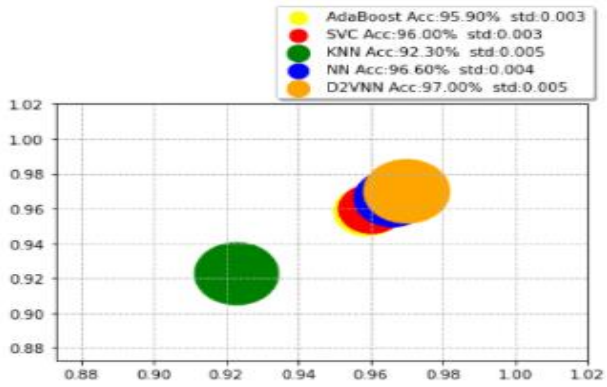
	precision	recall	f1-score	support
0	0.99	0.97	0.98	1555
1	0.97	0.99	0.98	1615
accuracy			0.98	3170
macro avg	0.98	0.98	0.98	3170
weighted avg	0.98	0.98	0.98	3170





Si noti come un embeddings di questo tipo, abbinato ad una buona rete neurale, permetta di ottenere risultati migliori. Come si evince da tutti i risultati e i grafici precedenti, il modello migliore per accuratezza è rappresentato dalla rete neurale con embeddings Doc2Vec, seguita da NN, SVC, KNN e AdaBoost ottenuti sulla prima tipologia di embeddings.

	accuracy_mean	std
AdaBoost	0.959	0.003
SVC	0.960	0.003
KNN	0.923	0.005
NN	0.966	0.004
D2VNN	0.970	0.005

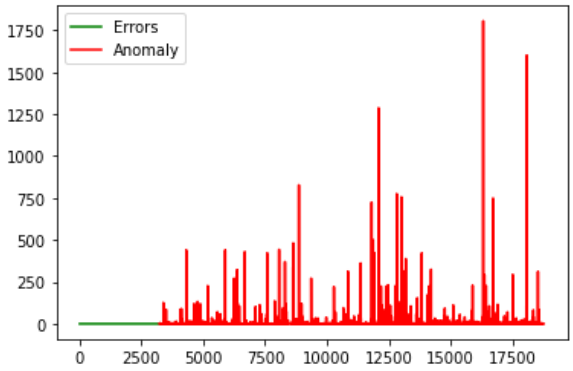


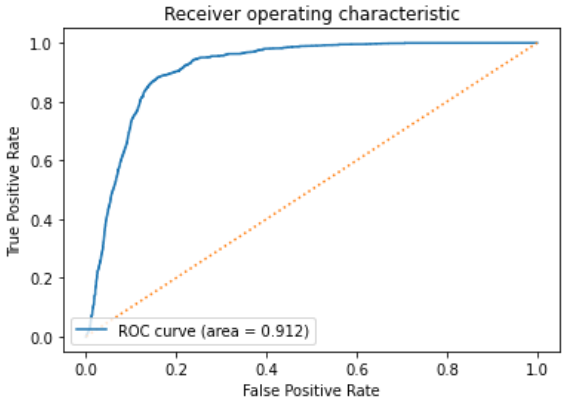
Anomaly Detection

Come richiesto dalla traccia progettuale, si cerca di identificare la classe più numerosa classificandola come corretta; le restanti classi rappresenteranno le anomalie.
Si mostra, tramite il grafico sottostante, che la classe ham è quella che contiene più esempi.

PCA

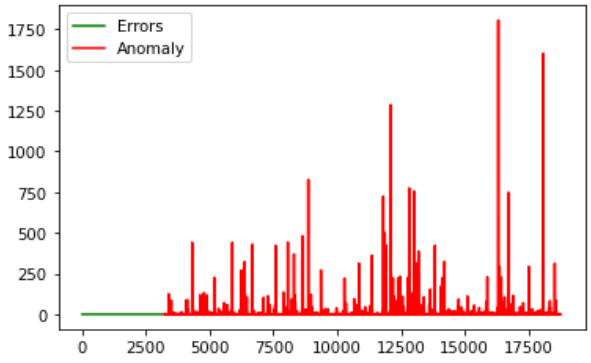
La migliore configurazione è con 256 features. Il modello non commette errore, quasi nullo sulla classe corretta, presentando un buon risultato. La curva di ROC conferma tutto questo, mostrando un risultato altamente accurato.

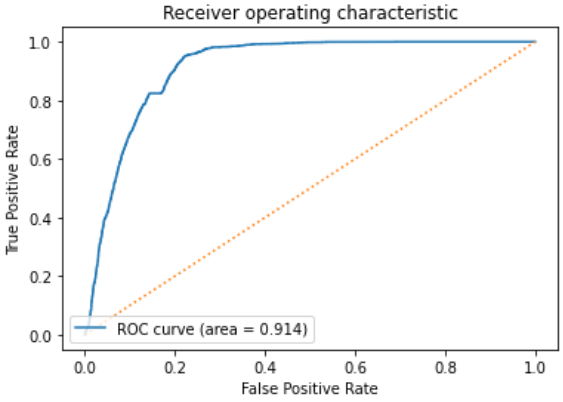




Si passa ora all'analisi PCA sulla rappresentazione tramite Doc2Vec. La migliore configurazione è con 278 features.

Anche in questo caso il modello non commette errore, quasi nullo, sulla classe corretta, presentando un buon risultato. La curva di ROC conferma tutto questo, mostrando un risultato altamente accurato, leggermente migliore di quello precedente.





Si salva il modello ottenuto dalla 2 configurazione.