Progetto Machine Learning

Giacomo Lancia
Fabio Peluso

28 settembre 2017

Capitolo 1

Introduzione

Il progetto presentato in questa relazione è relativo al dataset Statlog (Landsat Satellite), disponibile all'url `http://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29`.
I dati in esso contenuto rappresentano una immagine satellitare: si tratta di valori numerici associati a quadrati di 3×3 pixel vicini, classificati guardando il pixel centrale. Lo scopo del programma eseguito, realizzato in linguaggio Python utilizzando Jupyter Notebook, è quello di classificare ciascun quadrato di pixel in sette classi distinte, corrispondenti ad altrettante tipologie di suolo. L'obiettivo finale è dunque quello di poter ottenere dall'immagine una descrizione dettagliata e affidabile delle tipologie di terreno affinché i dati possano essere utilizzati per studi successivi. Le classi di terreno sono le seguenti (tra parentesi il numero della classe): red soil (1), cotton crop (2), grey soil (3), damp grey soil (4), soil with vegetation stubble (5), mixture class (6), very damp grey soil(7). Nella descrizione del DataSet viene affermato che la classe 6 non è presente nell'immagine. Prima di procedere con la classificazione, per la quale sono stati usati diversi classificatori per confrontarne le prestazioni, sono state effettuate delle operazioni di pre-processing. Al termine di tali operazioni abbiamo diviso il DataSet in Training Set e Test Set, riservando al primo il 75% dei dati e il restante 25% al secondo, procedendo con l'implementazione dei seguenti classificatori:

- Stochastic Gradient Descent;

- Support Vector Machine lineare;
- Support Vector Machine gaussiana;
- Support Vector Machine polinomiale di ordine 3;
- Logistic Regression;
- Gauss Naive Bayes;
- K Nearest Neighbors (Ball Tree);
- K Nearest Neighbors (KD Tree);
- AdaBoost.

Inoltre, per ciascun classificatore abbiamo eseguito il metodo della cross validation, per studiare eventuali fenomeni di overfitting e confrontare i risultati ottenuti.
Nel listato 1 si riporta per completezza l'header del programma con le librerie importate.

Listing 1.1: header del programma

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import sem
import seaborn as sns

from sklearn.svm import SVC #Classificatore SVC
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import cross_val_score, KFold
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn import linear_model
from sklearn import metrics
from sklearn import gaussian_process
from sklearn import naive_bayes
from sklearn import neighbors
from sklearn.ensemble import AdaBoostClassifier

from sklearn import datasets
```

Capitolo 2

Pre-processing

Nella fase di pre-processing sono state effettuate le operazioni preliminari sul DataSet.
In primo luogo, abbiamo convertito i target per eliminare dal conteggio la classe 6, non presente nel DataSet, e numerato le restanti sei classi da 0 a 5 (dato che Python numera gli indici di liste e array a partire da 0) con il seguente codice:

Listing 2.1: Eliminazione classe 6 e shift delle classi

```
for i in range (0,np.shape(target)[0]):
    target[i]=int(target[i])
    if target[i]==7:
    target[i]=6
target-=1
```

Successivamente abbiamo definito Training Set e Test Set assegnando ad essi rispettivamente il 75% e il 25% del DataSet:

Listing 2.2: Definizioni di Training Set e Test Set

```
X_train, X_test, y_train, y_test=
train_test_split(X, y, test_size=.25, random_state=66)
```

A questo punto è possibile riscalare i dati per ottere delle variabili a media nulla e varianza unitaria. Tale operazione può essere completata con i metodi `StandardScaler` e `.fit()`:

Listing 2.3: Codice per riscalare i dati dopo aver calcolato media e deviazione standard

```
scaler=preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

In alternativa si può applicare la PCA sui dati utilizzando il listato seguente:

Listing 2.4: PCA sui dati

```
estimator = PCA()
X_train = estimator.fit_transform(X_train)
X_test = estimator.fit_transform(X_test)
```

La serie di operazioni eseguite sui dati ci permette di semplificare, per quanto possibile, la complessità del DataSet. e rendere più veloci gli algoritmi usati. In figura 2.1 è riportato il plot bidimensionale del dataset manipolato con StandardScaler.

Figura 2.1: Plot del DataSet manipolato

Capitolo 3

Classificatori

Al fine di effettuare uno studio completo dei classificatori, lineari e non, si è ritenuto opportuno testare diversi metodi (elencati nell'introduzione) per poter analizzare la qualità della classificazione ottenuta e confrontare le differenti accuratezze risultanti. Si è dunque utilizzato un ciclo for per per effettuare training e valutazione, la cross validation e il plot di ciascun classificatore; il codice utilizzato è il seguente:

Listing 3.1: definizioni e chiamata ai classificatori

```
clf_set=
[linear_model.SGDClassifier(loss='squared_hinge',alpha = 1e-3),
SVC(kernel='linear'),
SVC(gamma=0.2, C=1), SVC(kernel='poly'),
linear_model.LogisticRegression(),
naive_bayes.GaussianNB(),
neighbors.KNeighborsClassifier(n_neighbors=4,
algorithm='ball_tree'),
neighbors.KNeighborsClassifier(n_neighbors=8,
algorithm='ball_tree'),
neighbors.KNeighborsClassifier(n_neighbors=4,
algorithm='kd_tree'),
neighbors.KNeighborsClassifier(n_neighbors=8,
algorithm='kd_tree'),
AdaBoostClassifier(n_estimators=100, learning_rate=0.6)]
```

```
16
17 #Labels per i classificatori
18 clf_patch=['SGDClassifier',
19 'SVC lineare',
20 'SVC gaussiana',
21 'SVC polinomiale (Ordine 3)',
22 'Regressione Logistica',
23 'Gaussian Naive Bayes',
24 'Nearest Neighbors (4, Ball tree)',
25 'Nearest Neighbors (8, Ball tree)',
26 'Nearest Neighbors (4,KD tree)',
27 'Nearest Neighbors (8,KD tree)',
28 'Adaboost']
29
30 # I dati da utilizzare
31 #(le due componenti principali per esempio)
32 x_train = X_train
33 x_test = X_test
```

Listing 3.2: codice per l'esecuzione del ciclo for

```
1 #Risultati di ogni classificazione e plot delle
2 #superfici di delimitazione
3 for patch_index in range (len(clf_set)):
4   train_and_evaluate(clf_set[patch_index],
5    x_train, x_test, y_train, y_test, patch_index)
6   evaluate_cross_validation (clf_set[patch_index],
7    x_train, y_train, 10, patch_index)
8   plot_boundary_decision(clf_set[patch_index],X_train,
9                                          y_train)
```

Listing 3.3: addestramento e accuratezza dei classificatori

```
1 def train_and_evaluate(clf,
2  X_train, X_test, y_train, y_test, patch_index):
3   print('Classifier:'+' '+clf_patch[patch_index])
```

```
4    clf.fit(X_train, y_train)
5
6    print ("Accuracy on training set:")
7    print (clf.score(X_train, y_train))
8    print ("Accuracy on testing set:")
9    print (clf.score(X_test, y_test))
10
11   y_pred = clf.predict(X_test)
12
13   print ("Classification Report:")
14   print (metrics.classification_report(y_test, y_pred))
15   print ("Confusion Matrix:")
16   print (metrics.confusion_matrix(y_test, y_pred))
17   print (' ')
```

Listing 3.4: K-fold cross validation

```
1  def evaluate_cross_validation(clf, X, y, K, patch_index):
2    print('K-FOLD CROSS VALIDATION: '+' '+clf_patch[patch_index])
3    print(' ')
4    # Creo un iteratore per la k-fold cross validation
5    cv = KFold(len(y), K, shuffle=True, random_state=0)
6    # In default il risultato \'e ottenuto dalla accuratezza.
7    scores = cross_val_score(clf, X, y, cv=cv)
8    print (scores)
9    print (("Mean score: {0:.3f} (+/-{1:.3f})").format(
10   np.mean(scores), sem(scores)))
11   print('**********************...')
12   print(' ')
```

3.1 Stochastic Gradient Descent

La prima classificazione è stata effettuata implementando lo Stochastic Gradient Descent (SGD) utilizzando il metodo **SGDClassifier()**, importato da `sklearn.linear_model`. Tale classificatore minimizza una funzione di costo data stimando il gradiente e cercando il minimo con un metodo iterativo.
La funzione di costo considerata si definisce nel modo seguente:

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

dove L può essere di varie forme (nel nostro caso minimi quadrati), R è una funzione di penalizzazione che può essere definita utilizzando la norma $\mathcal{L}^2$, la norma $\mathcal{L}^1$, oppure una opportuna combinazione convessa delle 2, α è un iperparametro non negativo.
Il metodo SGD approssima il gradiente di $E(w,b)$ utilizzando un solo elemento del training set per ogni iterazione e il suo funzionamento ottimale si ha su dati a media nulla e varianza unitaria giustificando in tal modo le operazioni preliminari sui dati descritte nella sezione dedicata al pre-rocessing. Con l'applicazione di tale metodo si ottengono i seguenti risultati:

Stochastic Gradient Descent con minimi quadrati				
class	precision	recall	f1-score	support
0	0.93	0.84	0.89	377
1	0.86	0.85	0.86	192
2	0.77	0.91	0.84	351
3	0.47	0.06	0.10	154
4	0.40	0.65	0.50	176
5	0.69	0.70	0.70	359
media/totale	0.73	0.73	0.71	1609

Si ottiene una accuratezza sul training set di 0.71, mentre l'accuratezza sul test set è di 0.73.
L'applicazione della K-Fold Cross Validation restituisce il seguente risultato:

```
[ 0.65010352  0.67908903  0.66252588  0.66252588  0.72256729
  0.69151139 0.7593361   0.72614108  0.71576763  0.69502075]
```

Mean score: 0.696 (+/-0.011).

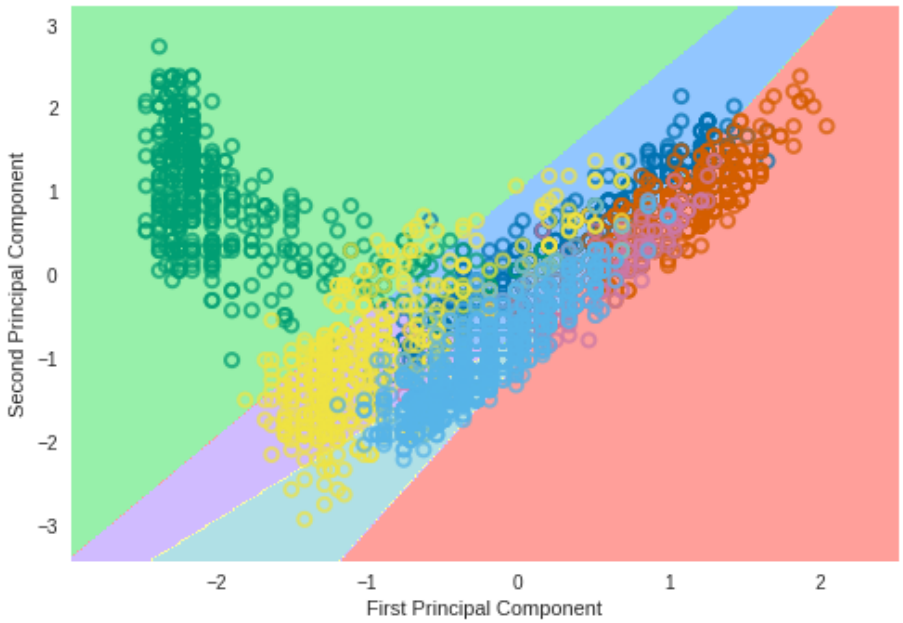

Figura 3.1: Classificazione con SGDClassifier

In figura 3.1 troviamo la rappresentazione della classificazione, dalla quale possiamo notare che una classe (la quarta) è assente, come si evince dalla tabella.

3.2 Support Vector Machines

In questa sezione viene descritta la tecnica delle Support Vector Machines, impletentata con il classificatore **SVC()**; esso è stato testato in tre modi diversi: con kernel lineare, con kernel Gaussiano e infine con kernel polinomiale di grado 3.

Una Support Vector Machine può fare classificazione multiclasse applicando un metodo *uno contro uno* ed utilizza un'idea di base relativamente semplice: si definiscono degli iperpiani in spazi di alta dimensione

(anche infinita) che separano i dati in regioni ben definite di spazio e si cerca di massimizzare la separazione di tali i dati con questi iperpiani. Teoricamente, la classificazione migliore si ottiene quando vengono definiti i margini funzionali, ovvero quando l'iperpiano di separazione ha distanza massima dai più vicini punti del training set per ogni classe. Per ciascuna coppia item-target (x_i, t_i) del training set, il margine funzionale rispetto alla direzione $\vec{w}$ e al parametro w_0 è definito da

$$\bar{\gamma}_i = t_i(\vec{w}^T \phi(x_i) + w_0),$$

mentre rispetto all'interno trainig set il margine funzionale è

$$\gamma = \min_i \bar{\gamma}_i.$$

Nel caso binario, che si estende al caso multiclasse con il meccanismo *1 contro 1*, il metodo SVC risolve il problema primale dato da

$$\begin{cases} \min_{w,b,\zeta} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \zeta_i \\ y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i \\ \zeta_i \geq 0 \quad \forall i \end{cases}.$$

Tale problema è equivalente al seguente, detto problema duale:

$$\begin{cases} \min_\alpha \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ y^T \alpha = 00 \leq \alpha_i \leq C, \forall i \end{cases}.$$

Nella formulazione di tali problemi, si considera

$$\begin{cases} x_i \in \mathbb{R}^p, i = 1, \ldots, n \\ y \in \{1, -1\}^n, e^T = (1, \ldots, 1) \\ C > 0 \text{ è un limite superiore} \\ (Q_{i,j})_{i=1,\ldots,n}^{j=1,\ldots,n} = y_i y_j \kappa(x_i, x_j) \text{ è una matrice } n \times n \text{ semidefinita positiva} \\ \kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j) \text{ è la funzione kernel} \end{cases}$$

Di seguito i risultati ottenuti.

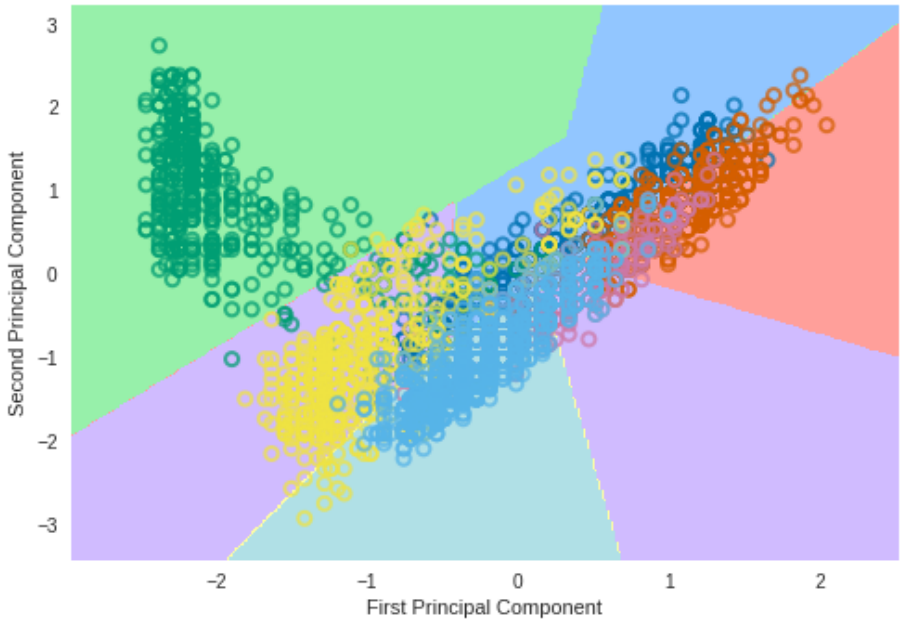

Figura 3.2: Classificazione con SVC e kernel lineare

SVC lineare

SVC con kernel lineare				
class	precision	recall	f1-score	support
0	0.90	0.96	0.93	377
1	0.97	0.88	0.92	192
2	0.86	0.92	0.89	351
3	0.59	0.42	0.49	154
4	0.86	0.73	0.79	176
5	0.79	0.88	0.83	359
media/totale	0.84	0.85	0.84	1609

Si ha una accuratezza sul training set di 0.844 e una accuratezza sul test set di 0.846. Il risultato della K-fold cross validation è

```
[ 0.85714286  0.81573499  0.83229814  0.83643892  0.83436853
  0.80331263 0.85684647  0.87136929  0.84232365  0.85892116]
```

Mean score: 0.841 (+/-0.007).

SVC Gaussiana

SVC gaussiana con gamma= 0.2e C= 1				
class	precision	recall	f1-score	support
0	0.91	0.96	0.94	377
1	0.99	0.88	0.93	192
2	0.85	0.93	0.89	351
3	0.58	0.43	0.49	154
4	0.90	0.78	0.84	176
5	0.80	0.87	0.83	359
media/totale	0.85	0.85	0.85	1609

L'accuratezza sul test set è 0.855, mentre sul test set è 0.854. Per la k-fold cross validation si ha

```
[ 0.85300207  0.83436853  0.82608696  0.8426501   0.83436853
  0.83229814 0.87966805  0.87136929  0.87344398  0.86929461]
```
Mean score: 0.852 (+/-0.006).

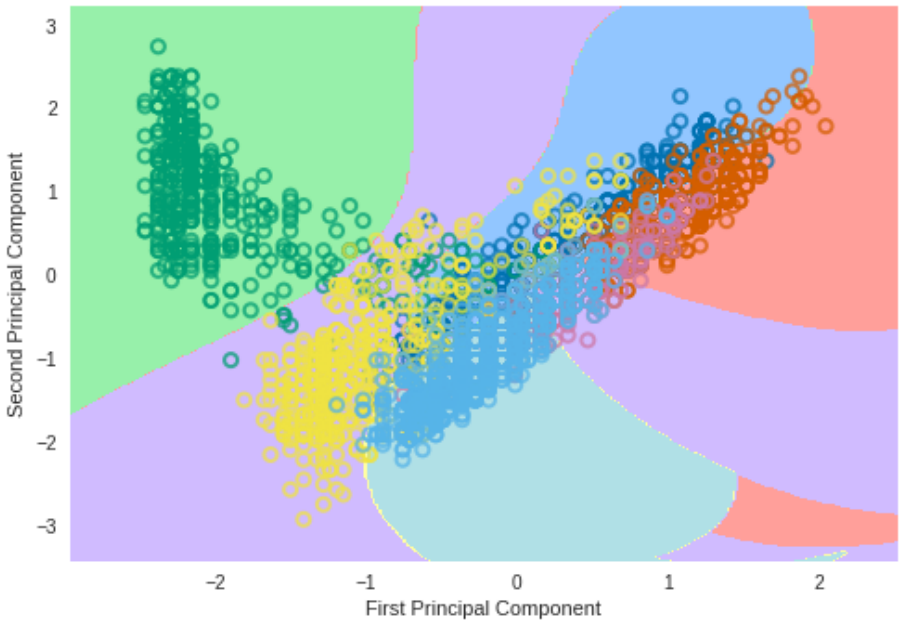

Figura 3.3: Classificazione con SVC Gaussiana

SVC polinomiale

SVC con kernel polinomiale di grado 3				
class	precision	recall	f1-score	support
0	0.82	0.98	0.90	377
1	0.99	0.87	0.93	192
2	0.89	0.88	0.88	351
3	0.50	0.51	0.50	154
4	0.94	0.66	0.78	176
5	0.82	0.83	0.82	359
media/totale	0.84	0.83	0.83	1609

Si ottiene una accuratezza sul training set di 0.826 e di 0.831 sul test set. Dalla k-flod cross validation si ha

```
[ 0.81780538  0.80952381  0.80538302  0.82194617  0.80952381
  0.80331263 0.83609959  0.83609959  0.84854772  0.84647303]
```

Mean score: 0.823 (+/-0.005).

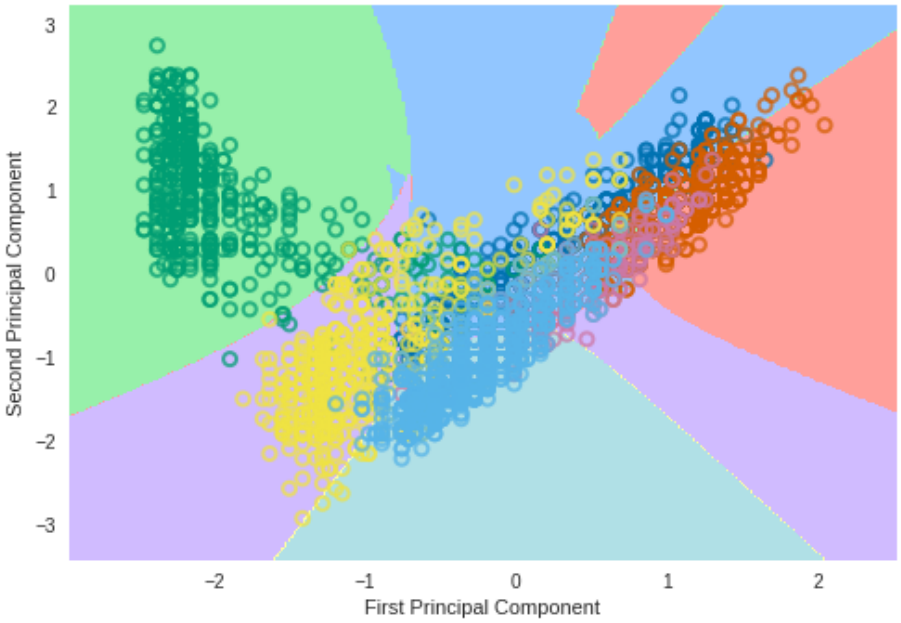

Figura 3.4: Classificazione con SVC e kernel polinomiale

3.3 Logistic Regression

Il metodo **LogisticRegression()** implementa un modello di classificazione probabilistico lineare che utilizza la *funzione logistica* per determinare i parametri della distribuzione di probabilità. Per semplicità riportiamo la formulazione del problema nel caso binario. Utilizzando una penalizzazione $\mathcal{L}^2$ il classificatore risolve il problema di ottimizzazione seguente:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1),$$

mentre nel caso con penalizzazione $\mathcal{L}^1$ il problema è dato da:

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Il metodo LogisticRegression() può utilizzare diversi solver: "liblinear", "newton-cg", "lbfgs", "sag" and "saga". Per utilizzare al meglio questo classificatore, è bene tener presente che "liblinear" implementa solo lo schema *uno contro tutti* e rappresenta una buona scelta per piccoli DataSet; per i problemi multiclasse la funzione di perdita multinomiale è gestita da "newton-cg", "sag", "saga" e "lbfgs"; "sag" e "saga" sono più veloci su DataSet grandi; infine "newton-cg", "lbfgs" e "sag" possono implementare solo la funzione di penalità $\mathcal{L}^2$.

Regressione logistica				
class	precision	recall	f1-score	support
0	0.88	0.96	0.92	377
1	0.89	0.89	0.89	192
2	0.81	0.96	0.88	351
3	0.25	0.01	0.01	154
4	0.91	0.47	0.62	176
5	0.68	0.93	0.79	359
media/totale	0.76	0.80	0.76	1609

L'accuratezza che si ottiene è 0.803 sul training set e 0.802 sul test set; per la k-fold cross validation si ha

```
[ 0.80124224  0.77639752  0.79917184  0.79710145  0.77432712
  0.76604555 0.81950207  0.83817427  0.81742739  0.83195021]
```

Mean score: 0.802 (+/-0.008).

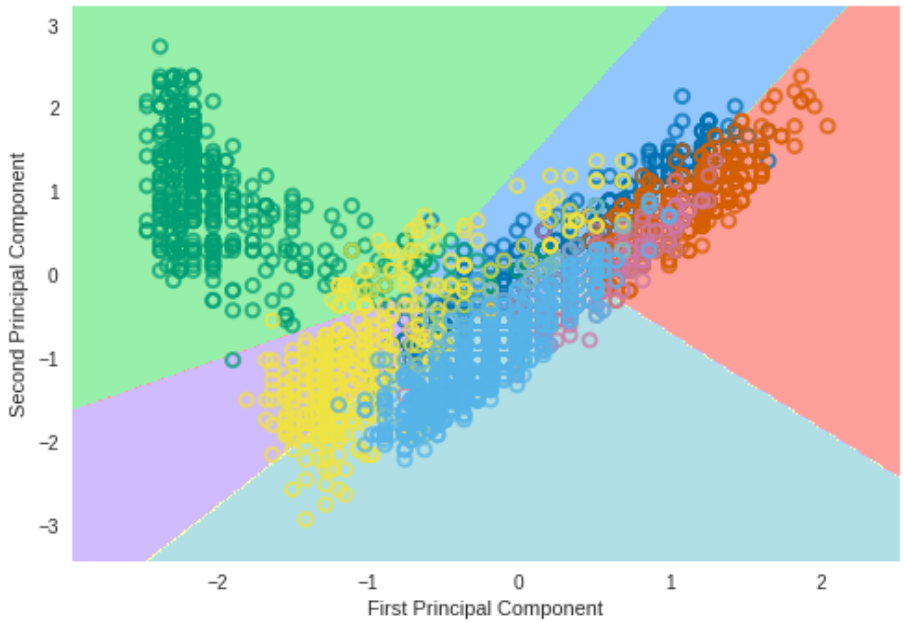

Figura 3.5: Classificazione con regressione logistica

3.4 Gauss Naive Bayes

Il metodo **GaussianNB()**, implementa l'algoritmo Gauss Naive Bayes, in cui si assume che la verosimiglianza sulle features sia Gaussiana:

$$\mathbb{P}(x_i|y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2} \right)$$

dove σ e μ sono stimati usando la massima verosimiglianza. L'aggettivo *naive* si riferisce all'ipotesi che le distribuzioni delle variabili input $x_1, \ldots, x_n$, condizionate rispetto alla classe C_k siano indipendenti.

Risultati del classificatore Gauss Naive Bayes				
class	precision	recall	f1-score	support
0	0.90	0.78	0.83	377
1	1.00	0.86	0.92	192
2	0.86	0.89	0.88	351
3	0.49	0.60	0.54	154
4	0.62	0.67	0.64	176
5	0.76	0.79	0.78	359
media/totale	0.80	0.79	0.79	1609

L'accuratezza ottenuta è di 0.787 sia sul training set che sul test set. Eseguendo la k-fold cross validation si ottiene

```
[ 0.7805383   0.76190476  0.76397516  0.80124224  0.77639752
  0.76397516 0.82157676  0.81327801  0.80082988  0.79875519]
```
Mean score: 0.788 (+/-0.007).

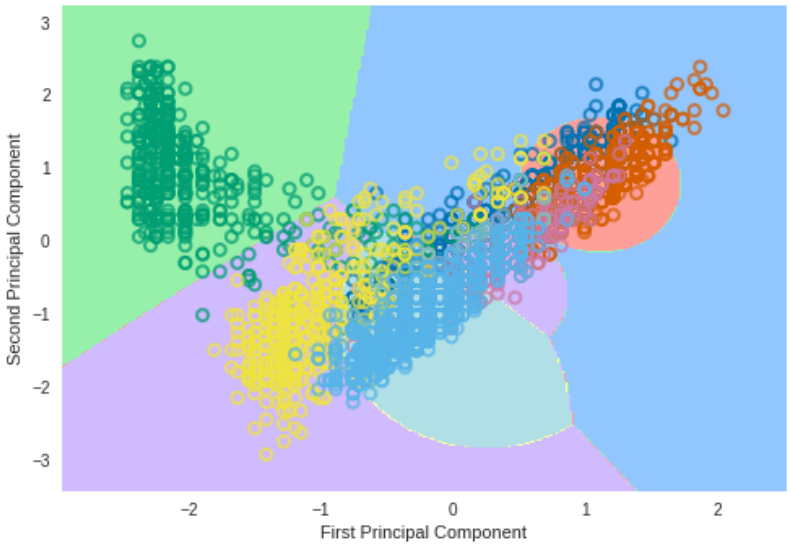

Figura 3.6: Classificazione con Naive Bayes gaussiano

3.5 Primi Vicini

La Nearest Neighbors Classification supervisionata, realizzata con il metodo **KNeighborsClassifier()** è un tipo di *instance-based learning*: infatti esso memorizza istanze del Training Set e classifica con un sistema di "voto a maggioranza", assegnando ad un dato punto la classe che conta più rappresentanti tra i punti vicini a quello dato.

KNeighboursClassifier() implementa un algoritmo di apprendimento specificando il numero k di primi vicini da considerare per ciascun punto. La scelta di k dipende fortemente dai dati: un numero molto elevato sopprime effetti di "rumore", ma i bordi decisionali risultano più difficili da distinguere, portando a un potenziale overfitting.

È possibile classificare i punti anche specificando per essi dei pesi definiti dall'attributo `weights`: esso può prendere il valore "uniform", che assegna un peso uniforme a tutti i punti, oppure "distance", che assegna a ciascun punto un peso inversamente proporzionale alla distanza dal punto che si sta classificando.
Per il metodo utilizzato sono disponibili diversi algoritmi per la soluzione del problema dei K primi vicini: sono implementati il "BallTree" (i dati vengono strutturati in ipersfere annidate), il "KDTree" (i dati sono organizzati in alberi binari i cui punti sono k-dimensionali), "brute" (per il quale si utilizza un metodo di brute-force), "auto" (che determina quale dei precedenti si adatti meglio al tipo di dati in esame).

Algoritmo BallTree, 4 punti

In questo caso si raggiunge una accuratezza di 0.886 sul training set e di 0.849 sul test set.
La k-fold cross validation ci dà

```
[ 0.83436853  0.81780538  0.83436853  0.8447205   0.81780538
  0.79503106 0.87344398  0.86099585  0.84024896  0.86099585]
```
Mean score: 0.838 (+/-0.007).

Primi vicini con BallTree, 4 punti				
class	precision	recall	f1-score	support
0	0.93	0.95	0.94	377
1	0.96	0.91	0.93	192
2	0.85	0.92	0.88	351
3	0.52	0.52	0.52	154
4	0.88	0.77	0.82	176
5	0.83	0.82	0.82	359
media/totale	0.85	0.85	0.85	1609

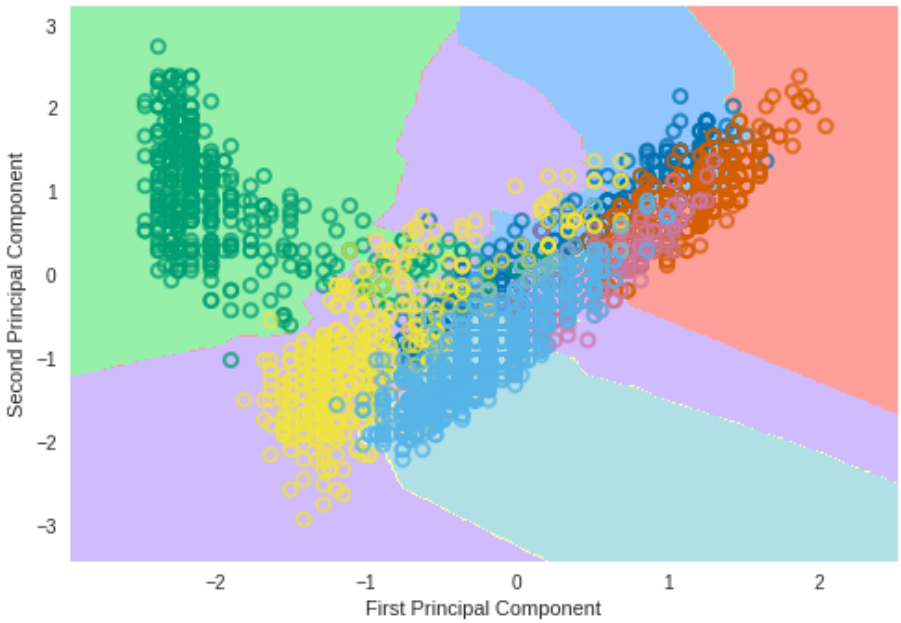

Figura 3.7: Classificazione con Nearest Neighbors e BallTree, 4 punti

Algoritmo BallTree, 8 punti

In questo caso è stato aumentato il numero di punti che viene considerato dall'algoritmo per risolvere il problema dei K punti. Si ottiene una accuratezza sul training set del 0.874, mentre sul test set è del 0.854.

Primi vicini con BallTree, 8 punti				
class	precision	recall	f1-score	support
0	0.92	0.97	0.95	377
1	0.99	0.89	0.93	192
2	0.87	0.92	0.90	351
3	0.54	0.55	0.54	154
4	0.89	0.77	0.83	176
5	0.82	0.82	0.82	359
media/totale	0.86	0.85	0.85	1609

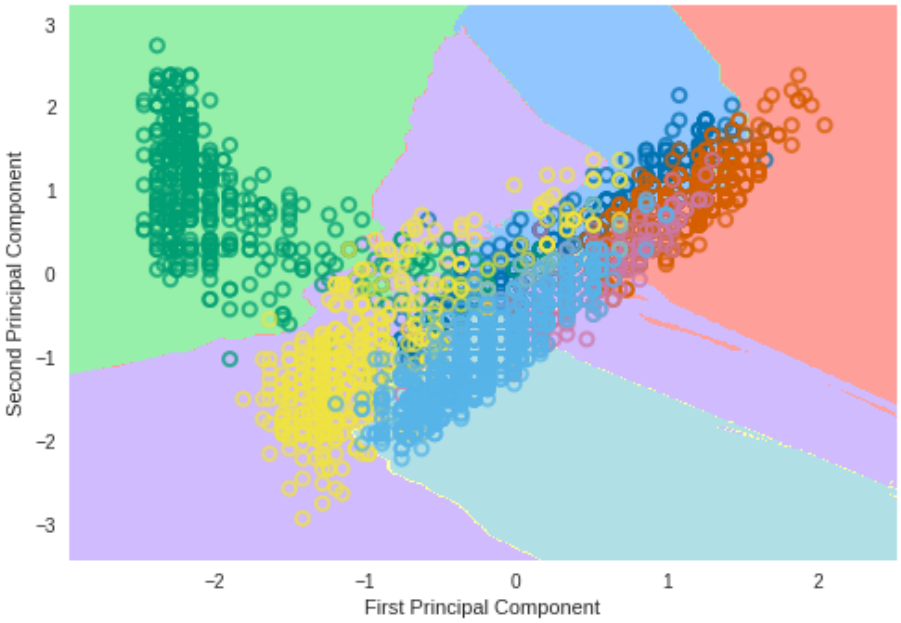

Figura 3.8: Classificazione con Nearest Neighbors e BallTree, 8 punti

In questo caso la k-fold cross validation ci dà

```
[ 0.84057971  0.82401656  0.84679089  0.83850932  0.82608696
  0.80952381  0.86929461  0.86514523  0.85684647  0.86721992]
```

Mean score: 0.844 (+/-0.006).

Algoritmo KDTree, 4 punti

Con algoritmo KDTree su 4 primi vicini otteniamo una accuratezza di 0.887 sul training set, mentre sul test sest otteniamo 0.849. Per la k-fold cross validation abbiamo

```
[ 0.83643892  0.8115942   0.83436853  0.84057971  0.81987578
  0.79710145 0.86929461  0.86099585  0.84024896  0.86099585]
```

Mean score: 0.837 (+/-0.007).

Primi vicini con KDTree, 4 punti				
class	precision	recall	f1-score	support
0	0.93	0.95	0.94	377
1	0.96	0.91	0.93	192
2	0.85	0.91	0.88	351
3	0.53	0.53	0.52	154
4	0.88	0.77	0.82	176
5	0.83	0.82	0.83	359
media/totale	0.85	0.85	0.85	1609

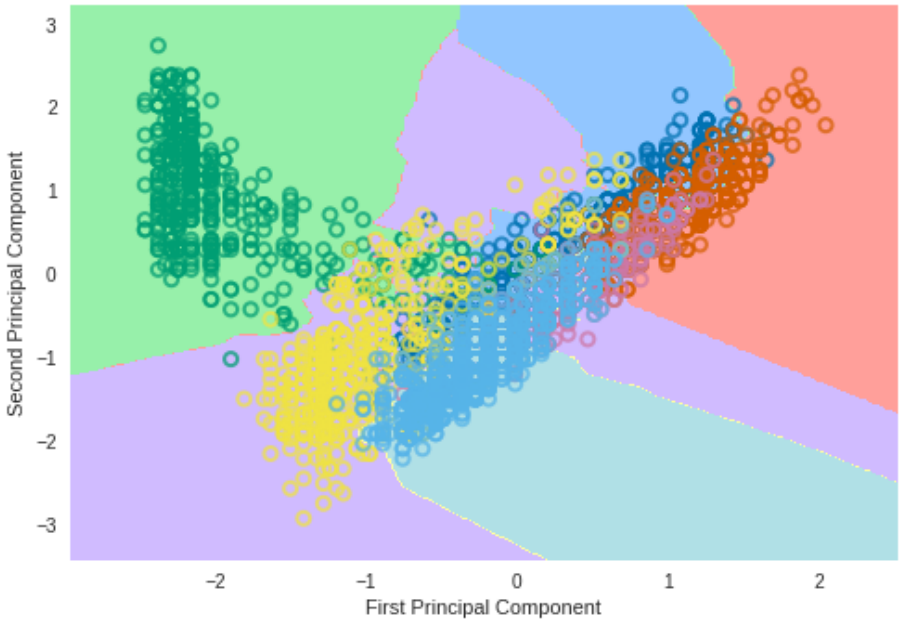

Figura 3.9: Classificazione con Nearest Neighbors e KDTree, 4 punti

Algoritmo KDTree, 8 punti

Aumentando il numero di punti vicini per l'algoritmo KDTree, si ottiene una accuratezza del 0.874 sul training set e del 0.858 sul test set. In questo caso per la k-fold cross validation si ha

```
[ 0.84057971  0.82608696  0.8426501   0.83850932  0.82815735
  0.8136646 0.87551867  0.86514523  0.85062241  0.86721992]
```

Mean score: 0.845 (+/-0.006).

Primi vicini con KDTree, 8 punti				
class	precision	recall	f1-score	support
0	0.93	0.97	0.95	377
1	0.99	0.89	0.93	192
2	0.87	0.93	0.90	351
3	0.55	0.55	0.55	154
4	0.89	0.77	0.83	176
5	0.82	0.84	0.82	359
media/totale	0.86	0.86	0.86	1609

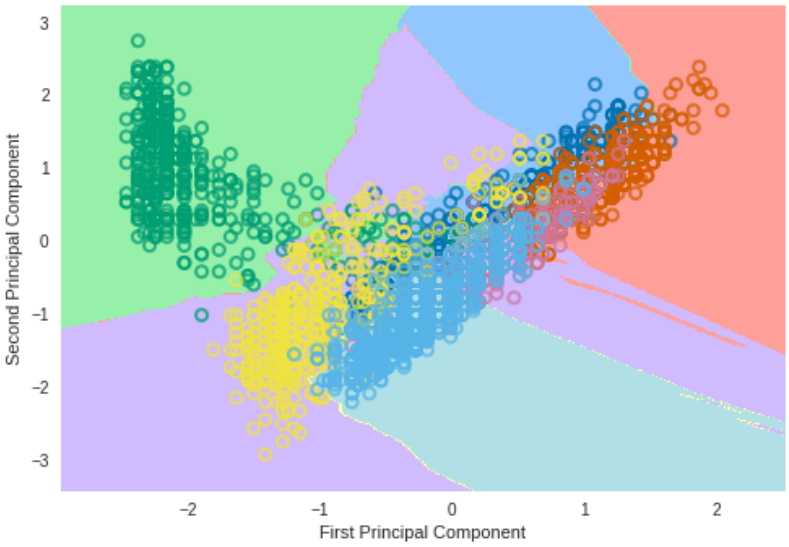

Figura 3.10: Classificazione con Nearest Neighbors e KDTree, 8 punti

3.6 AdaBoost

Il metodo **AdaBoostClassifier()** implementa il classificatore AdaBoost (adaptive boosting), il quale addestra in sequenza dei *weak learners* $G_i(x)$, $i = 1, \ldots, M$ (ad esempio *Decision Tree*) usando forme pesate del DataSet; i pesi vengono determinati a partire dalle performarce del run precedente e infine la predizione si ottiene con uno schema di voto a maggioranza: $G(x) = \text{sign}\Big(\sum_{i=1}^{M} \alpha_i G_i(x)\Big)$.

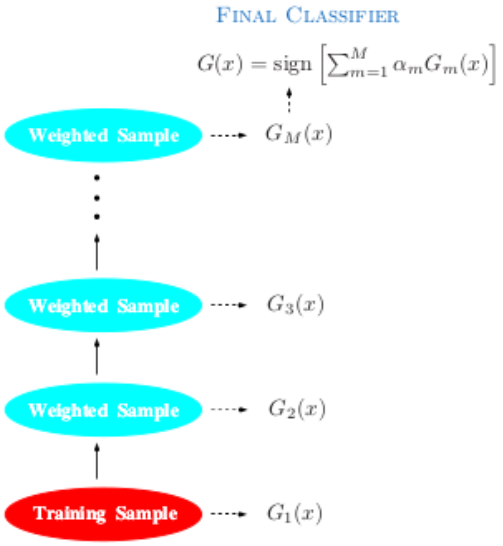

Figura 3.11: Schematizzazione tratta da *The Elements of Statistical Learning - Hastie, Tibshirani, Friedman - Springer.*

Adaboost con 100 stimatori e rate 0.6				
class	precision	recall	f1-score	support
0	0.86	0.85	0.85	377
1	0.78	0.79	0.79	192
2	0.82	0.90	0.86	351
3	0.47	0.52	0.50	154
4	0.74	0.47	0.57	176
5	0.76	0.80	0.78	359
media/totale	0.77	0.77	0.76	1609

L'accuratezza ottenuta sul training set è 0.792, mentre sul test set si ha 0.768. Per la k-fold cross validation troviamo

```
[ 0.74741201  0.73706004  0.59213251  0.42650104  0.75776398
  0.66666667 0.73236515  0.73236515  0.70539419  0.82365145]
```

Mean score: 0.692 (+/-0.035)

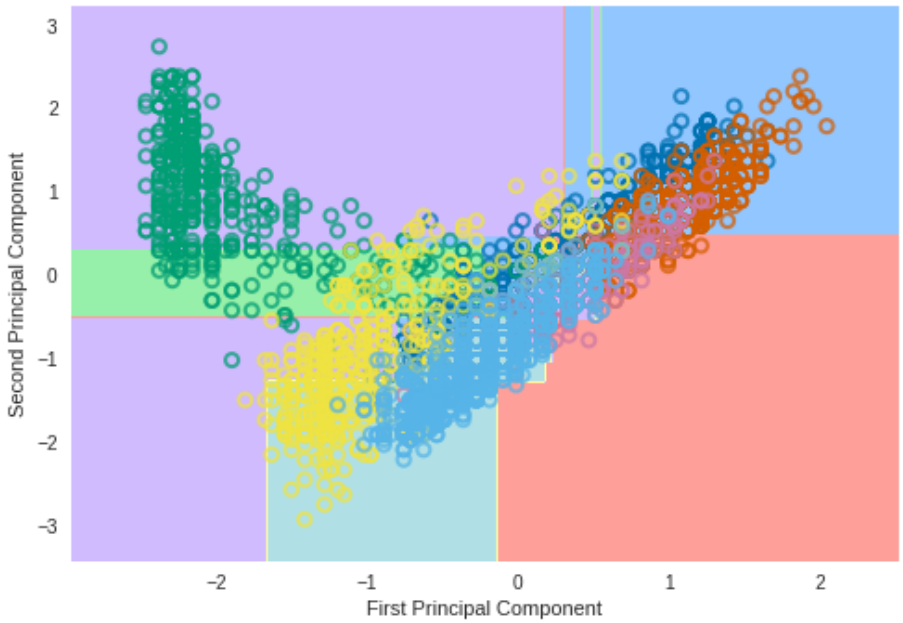

Figura 3.12: Classificazione con AdaBoost

Capitolo 4

Convalida Incrociata

Uno dei problemi più noti nei processi di classificazione è lo *overfitting*. Spesso si può riscontrare che la classificazione su un certo DataSet risulti estremamente accurata sul training set, ma non appena si inseriscono nuovi dati da classificare, tutte le predizioni vengono mancate. Quindi si ha un modello che descrive perfettamente i dati del training set ma che ha perso del tutto generalità. Per evitare la situazione appena descritta si ricorre all'utilizzo di metodi noti come *convalida incrociata*, o *cross validation*. Nel corso della nostra analisi, per ogni classificatore è stato utilizzato il metodo della **k-fold cross-validation**. La convalida incrociata consiste nella divisione dei dati del DataSet in partizioni uguali e nell'utilizzo di una di queste come training set per il classificatore, mentre la parte restante delle partizioni viene utilizzata come test set per verificare la validità delle predizioni del classificatore. Vengono eseguite tante analisi quante sono le partizioni. Il metodo utilizzato prevede la divisione casuale del DataSet in k (nel nostro caso 10) partizioni; di queste $k-1$ sono utilizzate per il training del modello e una come test. Un vantaggio evidente della *k-fold cross validation* è dato dal fatto che in questo modo il classificatore viene addestrato k volte su diversi insiemi di training ottenuti con una suddivisione casuale, e valutato k volte su partizioni ogni volta diverse. Al contrario, nella suddivisione data in partenza (75-25), il classificatore viene addestrato unicamente su una parte di dati e valutato sulla parte restante. La casualità della suddivisione in k *folds* e il successivo addestramento *a rotazione* limita la possibilità

di avere errori sistematici dovuti ad una cattiva suddivisione iniziale del DataSet. In questo modo, l'accuratezza media delle valutazioni ottenute dalla *k-fold cross validation* (Mean score) può essere confrontata con quella del modello testato in precedenza per darci una misura della bontà della classificazione. Il metodo restituisce anche l'ampiezza dell'intervallo di fiducia al 95%.

Esempio: Mean score 0.837 (+/-0.007) indica che la precisione media del metodo **KFold()** è di 0.837 e che l'ampiezza dell'intervallo di fiducia del 95% intorno a tale valore è di 0.007.

Capitolo 5

Conclusioni

L'analisi di questo DataSet ha permesso di mettere a confronto diversi metodi di classificazione. Tutti i metodi utilizzati hanno mostrato una buona accuratezza media. Tuttavia bisogna segnalare che l'accuratezza sulla classe 4, rinominata 3 in fase di pre-processing, risulta sempre molto lontana dalla media per ogni classificatore; in particolare, per tale classe, l'accuratezza è stata molto bassa per i classificatori **Stochastic Gradient Descent**, **Logistic Regression** e **AdaBoost** mentre la precisione massima è stata ottenuta con con **Support Vector lineare**, sebbene anche in quest'ultimo caso essa sia stata solo del 59%. Una così marcata *misclassification* potrebbe essere dovuta al tipo di dati presi in esame e potrebbe essere risolta con una ulteriore fase di preprocessing. Osserviamo che i risultati migliori sono stati ottenuti dai metodi **Support Vector Machines** e i metodi a **Primi Vicini**. In tabella 5.1 riportiamo un confronto tra i metodi usati.

Tabella 5.1: Riepilogo

Classificatore	Tipo	Accuratezza		CV Media
		Train	Test	
St. Gradient Descent	Minimi quadrati	0.714	0.732	0.696
Support Vector lineare		0.844	0.846	0.841
Support Vector gaussiana		0.855	0.854	0.852
Support Vector polinomiale	Grado 3	0.826	0.831	0.823
Logistic Regression		0.803	0.802	0.802
Gaussian Naive Bayes		0.787	0.787	0.788
K Nearest Neighbors	BallTree, 4	0.886	0.849	0.838
K Nearest Neighbors	BallTree, 8	0.874	0.854	0.844
K Nearest Neighbors	KDTree, 4	0.887	0.849	0.837
K Nearest Neighbors	KDTree, 8	0.874	0.858	0.845
AdaBoost	n=100, rate=0.6	0.792	0.768	0.692