Introducció

El present informe exposa els procediments i resultats d'execució de la **tercera pràctica** del curs d'Intel·ligència Artificial Avançada, relativa a la tasca de **classificació**.

El conjunt de dades que s'ha treballat s'hereta de la segona pràctica, i fa referència a les vendes d'una empresa distribuïdora d'aliments (*Wholesale Customers.csv*). L'arxiu inclou dues etiquetes (tipus de client i zona de procedència) i 6 dimensions, que corresponen al volum de vendes en m.u. de diferents tipus d'articles. L'objectiu de la pràctica és implementar un classificador que ens permeti **distingir els clients d'aquesta empresa a partir del volum de vendes en cada categoria**.

El fitxer de dades originals serà comú per a tots els apartats de la pràctica, i té la següent estructura:

Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_paper	Delicatessen

Les etiquetes prenen els següents valors (discrets):

Channel: 1: Hotel/Restaurant/Cafe (228 registres) – 2: Retail (142 registres)

Region: 1: Lisboa (77 registres) – 2: Oporto (47 registres) – 3: Altres (316 registres)

La resta de valors pot prendre qualsevol valor (continu) dins els següents rangs:

- FRESH: (3, 112151)
- MILK: (55, 73498)
- GROCERY: (3, 92780)
- FROZEN: (25, 60869)
- DETERGENTS_PAPER: (3, 40827)
- DELICATESSEN: (3, 47943)

Activitat 1

L'objectiu d'aquest exercici és la implementació d'un programa python que apliqui el Naïve Bayes, els arbres de decisió, el kNN i les màquines de vectors de suport (amb diferents kernels) a l'arxiu de dades "Wholesale customers.csv"; utilitzant la validació simple com a protocol de validació i l'error com a mesura d'avaluació.

La definició de classes serà la combinació de les dues primeres columnes de l'arxiu. És a dir, tindrem un problema amb sis classes que surten de totes les combinacions possibles de les dues classes: {Lisbon, Horeca}, {Lisbon, Retail}, {Oporto, Horeca}...

a. Tractament previ de les dades

Ta com vam fer a la pràctica 2, realitzarem un tractament previ de les dades que asseguri la independència de la contribució de la variable respecte a la seva magnitud potencial. D'altra manera, categories de venda propenses a arribar a rangs alts de valors, com la venda de productes frescos, tindrien major pes en el càlcul de distàncies o altres mètriques que d'altres categories com la venda de detergents.

Com ja vam veure, totes les variables representen volums de vendes per un mateix canal i presenten distribucions de probabilitat similars (veure figures pràctica 2 per a més detall). Entenent, per tant, que no cal ajustar la distribució de probabilitat de la variable en qüestió, decidim realitzar únicament un Escalat 0-1 de les dades per tal d'ajustar els rangs.

Recalcularem cada esdeveniment de cada variable del conjunt de dades mitjançant la següent conversió:

$$X' = \frac{X - 1}{\max(X) - 1}.$$

A diferència de dues les pràctiques anteriors, utilitzarem el mètode *preprocessing.normalize* de la biblioteca scikit-learn per a realitzar aquesta tasca, en detriment del càlcul manual. La utilització de mètodes de sklearn serà una constant durant tota la pràctica, i respon a l'objectiu personal d'aprofundir en les funcionalitats d'aquesta biblioteca.

La següent imatge recull l'importació de tots els mòduls necessaris per a la realització dels 3 apartats de la pràctica i la càrrega i preparació de les dades:

```
"""
WHOLESALES CUSTOMERS
Learning algorithm: kNN / Naïve Bayes / Decision Tree / SVM (multiple kernels)
With variable normalization
No Dimensionality reductionsingle-validation
Training set size = 4 x test size
No statistical test
"""

from random import shuffle
import numpy
import math
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree

from sklearn import metrics
from sklearn.externals.six import StringIO

from sklearn import preprocessing

import matplotlib.pyplot as plt

# open file and load data into a list of lists
l = list(map(lambda l: (l.strip()).split(','),
             open('Data/Wholesale Customers.csv', 'r').readlines()))

# delete head
del(l[0])

# examples shuffle
shuffle(l)

# Split thelist in Labels + Data
data = []
target = []
for x in l:
    data.append(x[2:])
    target.append(x[0] + x[1])
    # target.append(x[0])

X = numpy.array(data)
y = numpy.array(target)

X = X.astype(numpy.float)
X = preprocessing.normalize(X)

y = y.astype(numpy.float)
```

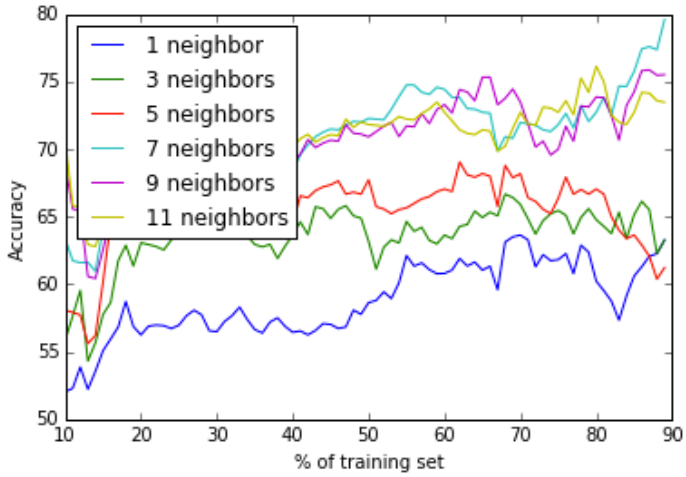
b. k-Nearest Neighbors

Per tal de ajustar els paràmetres fonamentals del kNN, hem realitzat un anàlisi del seu comportament amb valors de [1, 3, 5, 7, 9] veïns més propers (*k*) i amb una mida del conjunt d'entrenament d'entre el 10% i el 90% del volum total del conjunt de dades.

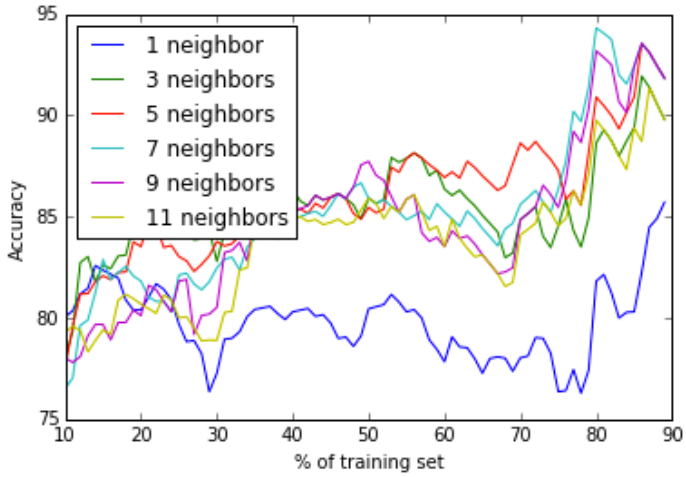
Les següents figures recullen la crida al mètode *neighbors.KNeighborsClassifier* de la biblioteca scikit-learn. Aquest mètode permet l'ajust de diversos paràmetres com el nombre de veïns a utilitzar per a la classificació (k), la funció de pes a utilitzar en la predicció (pesos uniformes de tots els veïns, pesos basculats segons la distància del punt al veí o funcions definides per l'usuari), l'algoritme utilitzat per a computar els veïns més propers o la mètrica de distància a utilitzar.

```
array = range(10, 90) # pct of the training set over the total dataset

pack =[]
neighbours = [1, 3, 5, 7, 9, 11]

labels = ("1 neighbor", "3 neighbors",
              "5 neighbors", "7 neighbors", "9 neighbors", "11 neighbors")

for i in array:
    accuracy = []

    # Train a k Nearest Neighbours Classifier
    for k in neighbours:

        # train_n = 1/float(i)
        train_n = float(i) / 100

        train_size = math.floor(len(l) * train_n)

        train_features = X[:train_size]
        train_labels = y[:train_size]

        test_features = X[train_size:]
        test_labels = y[train_size:]

        knn = KNeighborsClassifier(n_neighbors=k)
        y_pred = knn.fit(train_features, train_labels).predict(test_features)
        acc = 100 * knn.score(test_features, test_labels)

        # Query the accuracy of the model with the test set
        accuracy.append(acc)

    pack.append(accuracy)

P = numpy.array(pack)

plt.xlabel('% of training set')
plt.ylabel('Accuracy')

for i, l in zip(range(6), labels):
    print l
    plt.plot(array, P[:, i], label=l)

plt.legend(loc='upper left')
plt.show()
```

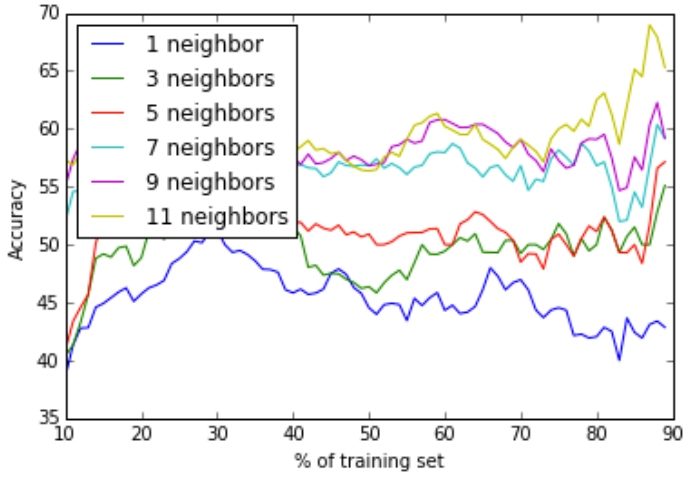
Predicció de la variable “regió”, segons el nombre de veïns utilitzats i el % de dades utilitzades en l’entrenament



Predicció de la variable “canal”, segons el nombre de veïns utilitzats i el % de dades utilitzades en l’entrenament



Predicció d’ambdues variables conjuntament:



Notem aquí que, com era d'esperar, **la precisió de l'estimació de l'etiqueta "canal" és molt superior que la predicció de l'etiqueta "regió"** (valors del 90% de la primera, vers valors del 70% de la segona): **tal i com vam veure a la segona pràctica, la variable "canal" explicava prou bé la distribució de les dades, però no així la variable "regió".** La precisió de la predicció d'ambdues variables conjuntament sembla aproximar-se al producte d'ambdues, la qual cosa resultaria lògica entenent la distribució d'aquesta com a combinació d'ambdues distribucions independents.

Com es pot veure a les figures, en cap dels 2 cassos hi ha un guany de precisió notable amb mides del conjunt d'entrenament superiors al 50% del volum total de dades. Si es nota **certa inestabilitat amb volums petits del conjunt d'entrenament, potser per la falta d'individus que puguin cobrir l'espai de dades amb una densitat suficient**, que asseguri el bon funcionament del classificador. Addicionalment, es nota que en la predicció de l'etiqueta "canal", augmentar el nombre de veïns necessaris per a fer l'etiquetat no augmenta notablement la precisió del classificador. En el cas de la predicció de l'etiqueta "zona", o d'ambdós conjuntament, al augmentar el valor de k augmenta la precisió del classificador, fins a estabilitzar-se amb valors de k propers a 7. Per aquest motiu, fixarem el volum del conjunt d'entrenament en el 60% del volum total de dades i el nombre de veïns necessaris per a l'etiquetat a 7.

Amb aquests valors, s'executa el codi de la figura següent per a obtenir els resultats per a la predicció d'ambdues classes mostrats a continuació:

```
# Train a kNN Classifier

train_n = 0.6
train_size = math.floor(len(l) * train_n)

train_features = X[:train_size]
train_labels = y[:train_size]

test_features = X[train_size:]
test_labels = y[train_size:]

k = 7

knn = KNeighborsClassifier(n_neighbors=7)
y_pred = knn.fit(train_features, train_labels).predict(test_features)
acc = 100 * knn.score(test_features, test_labels)
# single-validation
print("kNN Single-validation")
print("%d Neighbours. Size = %d pct of dataset. Number of mislabeled points out of a total %d points : %d"
            % (k, train_n * 100, test_features.shape[0],(test_labels != y_pred).sum()))
print("Accuracy: %d pct"
            % (acc))
```

Mètode: **kNN**

- Volum del conjunt d'entrenament: 60% del total (176 individus)
- Nombre d'errors: 75/176
- Error: 43%

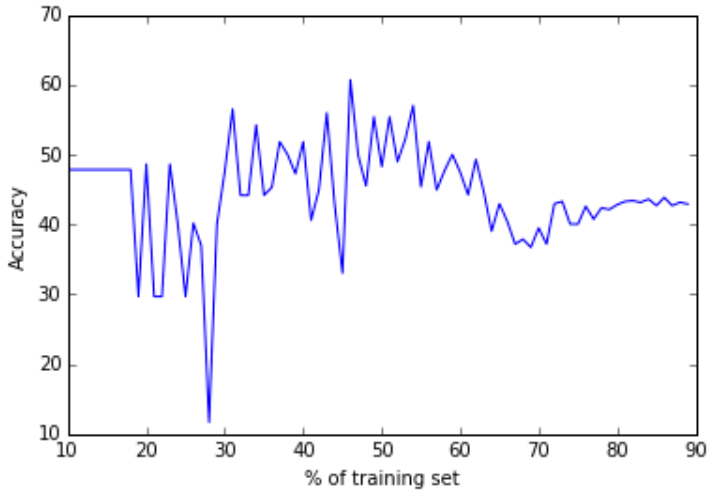
c. Decision Tree

Per a l'utilització d'aquest classificador hem tornat a recórrer a la llibreria scikit-learn, en aquest cas a través del mètode *tree.DecisionTreeClassifier*, que permet la crida al mètode amb diferents criteris de profunditat i de nombre d'exemples mínim necessaris per a construir una branca. De nou, hem avaluat la precisió del mètode amb diferents mides del conjunt

d'entrenament. Els valors dels paràmetres s'han mantingut per defecte (1 exemple val per a fer un node, sense profunditat màxima de l'arbre). El codi utilitzat i el resultat es mostren a continuació:

```
# Eval a decision tree
array = range(10, 90) # pct of the training set over the total dataset
pack =[]
for i in array:
    accuracy = []

    train_n = float(i) / 100

    train_size = math.floor(len(l) * train_n)

    train_features = X[:train_size]
    train_labels = y[:train_size]

    test_features = X[train_size:]
    test_labels = y[train_size:]

    dt = tree.DecisionTreeClassifier()
    y_pred = dt.fit(train_features, train_labels).predict(test_features)
    acc = 100 * dt.score(test_features, test_labels)

    # Query the accuracy of the model with the test set
    accuracy.append(acc)

    pack.append(accuracy)

P = numpy.array(pack)

plt.xlabel('% of training set')
plt.ylabel('Accuracy')

for i, l in zip(range(6), labels):
    print l
    plt.plot(array, P[:, i], label=l)

plt.legend(loc='upper left')
plt.show()
```

Predicció de la precisió, amb tots els valors per defecte:



Com es pot veure, la precisió obtinguda es similar a la del mètode kNN, i també **s'estabilitza per a mides del conjunt d'entrenament superiors al 50% del volum total de dades**. En aquest cas, però, sembla que el mètode presenta major inestabilitat per a volums petits del conjunt d'entrenament, probablement per la creació de branques amb pocs individus. Aquest comportament es podria ajustar amb un bon criteri de poda.

Els resultats obtinguts per a la predicció d'ambdues classes amb un volum del conjunt d'entrenament del 60% del total han sigut els següents:

```
158 # Train a Decission Tree
159
160 train_n = 0.6
161 train_size = math.floor(len(l) * train_n)
162
163 train_features = X[:train_size]
164 train_labels = y[:train_size]
165
166 test_features = X[train_size:]
167 test_labels = y[train_size:]
168
169 dt = tree.DecisionTreeClassifier()
170 y_pred = dt.fit(train_features, train_labels).predict(test_features)
171
172 # Query the accuracy of the model with the test set
173 acc = 100 * dt.score(test_features, test_labels)
174
175 # single-validation
176 print("Decision Tree Single-validation")
177 print("Number of mislabeled points out of a total %d points : %d"
178     % (test_features.shape[0],(test_labels != y_pred).sum()))
179 print("Accuracy: %d pct"
180     % (acc))
181
182 with open("Wholesale.dot", 'w') as f:
183     f = tree.export_graphviz(dt, out_file=f)
184
```

Mètode: Decision Tree

- Volum del conjunt d'entrenament: 60% del total (176 individus)
- Nombre d'errors: 99/176
- Error: 57%

L'arxiu Wholesale.dot, traduït a Wholesale.pdf, conté el conjunt de regles generades per l'arbre.

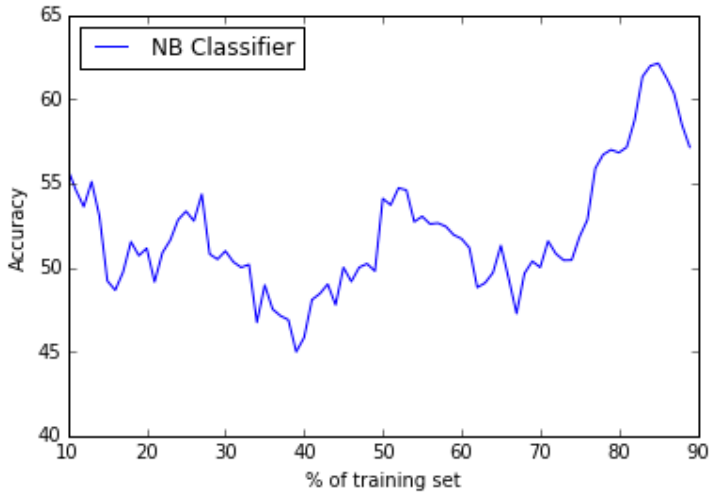
d. Naïve Bayes

Hem utilitzat el mètode *naive_bayes.GaussianNB*, de la llibreria scikit-learn, que implementa el classificador bayesià ingenu (no realitza presumpcions sobre la distribució de probabilitat de les variables a excepció de la seva independència). Aquest mètode permet l'ajust del nombre mínim d'exemples per a realitzar l'assignació a una classe.

De nou, s'ha avaluat el comportament del mètode segons la mida del conjunt d'entrenament. El resultat obtingut ha sigut el d'una **major dependència del nombre de dades per a realitzar una classificació acurada, potser relacionat amb el balanceig del conjunt d'entrenament** (mateix nombre de dades a cada classe). El codi utilitzat i els resultats obtinguts es mostren a continuació:

```
# Eval a Naive Bayes Classifier
array = range(10, 90) # pct of the training set over the total dataset
pack =[]
for i in array:
    accuracy = []

    train_n = float(i) / 100

    train_size = math.floor(len(l) * train_n)

    train_features = X[:train_size]
    train_labels = y[:train_size]

    test_features = X[train_size:]
    test_labels = y[train_size:]

    nb = GaussianNB()
    y_pred = nb.fit(train_features, train_labels).predict(test_features)
    acc = 100 * nb.score(test_features, test_labels)

    # Query the accuracy of the model with the test set
    accuracy.append(acc)

    pack.append(accuracy)

P = numpy.array(pack)

plt.xlabel('% of training set')
plt.ylabel('Accuracy')
plt.plot(array, P, label="NB Classifier")
plt.legend(loc='upper left')
plt.show()
```

Predicció de la precisió, amb tots els valors per defecte:



Els resultats obtinguts per a la predicció d'ambdues classes amb un volum del conjunt d'entrenament del 60% del total han sigut els següents:

Mètode: Naive Bayes

- Volum del conjunt d'entrenament: 60% del total (176 individus)
- Nombre d'errors: 75/176
- Error: 43%

e. Support Vector Machines

En aquest apartat utilitzarem màquines de vectors de suport amb diferents tipus de kernels, que ens permetran projectar les dades en diferents espais d'atributs : ineals, radials i polinòmics de grau 3. Atenent a la variabilitat de l'algorisme amb canvis d'escala, en aquest punt és especialment important realitzar la normalització de les dades presentada a l'apartat *a*. Per a realitzar la classificació, utilitzarem les crides a la llibreria *libsvm* implementada a la llibreria *scikit-learn,* amb paràmetre de penalització del terme d'error = 1 (valor per defecte), grau de polinomi = 3 pel cas del kernel polinòmic i gamma = 0 pel cas del kernel radial. El mètode de la llibreria scikit-learn permet l'ajust de molts altres paràmetres, alguns dels quals queden més enllà de l'abast d'aquesta assignatura.

```
# Train a SVM Clasifier
train_n = 0.6
train_size = math.floor(len(l) * train_n)

train_features = X[:train_size]
train_labels = y[:train_size]

test_features = X[train_size:]
test_labels = y[train_size:]

linear_svc = svm.SVC(kernel="linear")
y_pred = linear_svc.fit(train_features, train_labels).predict(test_features)
acc = 100 * linear_svc.score(test_features, test_labels)
# single-validation
print("SVM with linear kernel Single-validation")
print("Number of mislabeled points out of a total %d points : %d"
    % (test_features.shape[0],(test_labels != y_pred).sum()))
print("Accuracy: %d pct"
    % (acc))

rbf_svc = svm.SVC(kernel="rbf")
y_pred = rbf_svc.fit(train_features, train_labels).predict(test_features)
acc = 100 * rbf_svc.score(test_features, test_labels)
# single-validation
print("SVM with radial kernel Single-validation")
print("Number of mislabeled points out of a total %d points : %d"
    % (test_features.shape[0],(test_labels != y_pred).sum()))
print("Accuracy: %d pct"
    % (acc))

poly_svc = svm.SVC(kernel="poly")
y_pred = poly_svc.fit(train_features, train_labels).predict(test_features)
acc = 100 * poly_svc.score(test_features, test_labels)
# single-validation
print("SVM with Polynomic kernel Single-validation")
print("Number of mislabeled points out of a total %d points : %d"
    % (test_features.shape[0],(test_labels != y_pred).sum()))
print("Accuracy: %d pct"
    % (acc))

```

Els resultats obtinguts per a la predicció d'ambdues classes amb un volum del conjunt d'entrenament del 60% del total han sigut els següents:

Mètode: SVM - Kernel Lineal

- Volum del conjunt d'entrenament: 60% del total (176 individus)
- Nombre d'errors: 68/176
- Error: 39%

Mètode: SVM - Kernel Radial

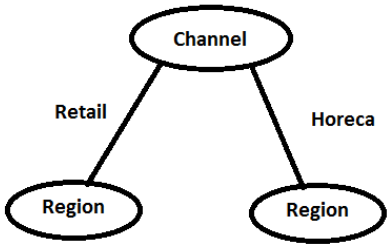
- Volum del conjunt d'entrenament: 60% del total (176 individus)
- Nombre d'errors: 70/176
- Error: 40%

Mètode: SVM - Kernel Polinòmic de grau 3

- Volum del conjunt d'entrenament: 60% del total (176 individus)
- Nombre d'errors: 88/176
- Error: 50%

Activitat 2

Una aproximació diferent a la de l'exercici anterior és fer servir una classificació en dues fases. És a dir, primer muntem un classificació per a distingir una de les classe i després un classificador per cadascun dels valors d'aquesta classe en un segon nivell (sabent que ja són de la classe); tal i com mostra la figura següent:



O a la inversa, tenint un classificador per Region en el primer nivell i tres per Channel en el segon.

Repetiu l'exercici anterior seguint ara aquest model de tractament jeràrquic de les classes.

En aquest cas, tal i com planteja l'enunciat, realitzarem una aproximació al problema en dues fases, classificant primer el canal i després la regió. L'algorisme a utilitzar serà el següent:

1. Carreguem el conjunt de dades, i realitzem el tractament previ necessari (normalització i mesclat de les dades).
2. Preparem un conjunt d'entrenament amb les variables de categories de compra. Així mateix, preparem 2 conjunts d'etiquetes: un amb les etiquetes de canal, i un altre amb les etiquetes de regió. Aquestes dues etiquetes poden estar dins una llista, la qual cosa agilitzarà el treball amb classificadors que permetin la predicció de llistes (kNN i Decision Tree). En cas contrari (Naïve Bayes i SVM) només haurem de separar la llista en dos literals.
3. Per a cadascun dels classificadors, entrenem el conjunt de dades (*fit*) i realitzem la predicció de l'etiqueta "canal" i de l'etiqueta regió, de manera independent. Amb les prediccions correctes de la primera (només amb aquestes), valorem la predicció de la segona. El volum d'encerts que surtin d'aquesta segona predicció, entre el nombre total d'individus del conjunt de test, donarà l'*accuracy* del mètode. Podem calcular l'error com la inversa d'aquest.

El codi següent recull el funcionament del classificador en dues fases pel cas de l'algorisme kNN (volum del conjunt d'entrenament del 60% i k=7). La resta d'algorismes funcionen d'una manera molt similar. La implementació de tots ells es pot trobar a l'arxiu *ActivityTwoDef.py* dins la carpeta de la pràctica.

```
62 # Train a kNN Classifier
63
64 k = 7
65
66 knn = KNeighborsClassifier(n_neighbors=7)
67 knn.fit(train_features, train_labels)
68 y_pred = knn.predict(test_features)
69
70 n_test_labels = 0.0
71 first_order_acc = 0.0
72 second_order_acc = 0.0
73
74 for pred, real in zip(y_pred, test_labels):
75     n_test_labels = n_test_labels + 1
76     if pred[0] == real[0]:
77         first_order_acc = first_order_acc + 1
78         if pred[1] == real[1]:
79             second_order_acc = second_order_acc + 1
80
81 first_order_error = 100 * (n_test_labels - first_order_acc) / n_test_labels
82 total_error = 100 * (n_test_labels - second_order_acc) / n_test_labels
83
84 # single-validation
85 print("kNN Single-validation")
86 print("%d Neighbours. Size = %d pct of dataset."
87             % (k, train_n * 100))
88 print("First step Error. %d pct. Second step Error: %d pct"
89             % (first_order_error, total_error))
90
```

Els resultats obtinguts amb l'aplicació dels diferents algorismes es recullen a continuació:

Mètode: kNN

- Error en primera instància (predicció del canal): 13%
- Error total (predicció conjunta): 46%

Mètode: Decision Tree

- Error en primera instància (predicció del canal): 22%
- Error total (predicció conjunta): 57%

Mètode: Naive Bayes

- Error en primera instància (predicció del canal): 14%
- Error total (predicció conjunta): 53%

Mètode: SVM Lineal

- Error en primera instància (predicció del canal): 11%
- Error total (predicció conjunta): 40%

Mètode: SVM Radial

- Error en primera instància (predicció del canal): 12%
- Error total (predicció conjunta): 40%

Mètode: SVM Polinòmic (grau 3)

- Error en primera instància (predicció del canal): 35%
- Error total (predicció conjunta): 57%

Exercici 3

Doneu una taula amb tots els resultats dels exercicis anteriors. Realitzeu una valoració global comparant els mètodes i redacteu unes conclusions globals sobre l'aplicació dels mètodes a aquest conjunt de dades. Els criteris de correcció de la PAC invaliden una A si tots els processos no estan ben justificats i comentats.

Mètode	Error – Classificació conjunta	Error – Classificació en 2 passos
k-Nearest Neighbors (k=7)	43%	46%
Decision Tree	57%	57%
Naïve Bayes	43%	53%
SVM – Lineal	39%	40%
SVM – Radial	40%	40%
SVM – Polinòmic 3	50%	57%

Conclusió i comentaris

En aquesta PAC s'han utilitzat diferents classificadors per a la realització d'una mateixa tasca (el reconeixement d'atributs de clients a partir del volum de vendes), a fi i efecte de poder extreure algunes conclusions relatives a les característiques i el comportament d'aquests:

- L'algorisme kNN ha demostrat un bon comportament en la tasca de predicció amb un bon rendiment computacional (0.181s). Cap destacar, com a virtud de l'algorisme, la seva simplicitat (sobretot en front d'altres com el SVM) i la seva expressivitat, el que permet l'ajust del mateix per tal de millorar el seu rendiment. Pel contrari, presenta l'inconvenient de ser un algorisme d'aprenentatge retardat: és a dir, no genera model d'aprenentatge, sinó que aquest succeeix en el mateix moment de la classificació i, per tant, s'han d'emmagatzemar totes les dades per tal de poder utilitzar-lo. Això pot presentar problemes en cas de treballar amb gran conjunts de dades, si bé hi ha variacions del kNN que resolen aquesta qüestió. A més, s'ha mostrat sensible al valor de k, la qual cosa pot obligar a realitzar processos de proves per tal de fixar aquest paràmetre.
- El Decission Tree presenta, com a gran virtut, la possibilitat de representar gràficament l'aprenentatge en forma d'arbre de decisió, amb un conjunt de regles fàcilment comprensibles que ens permeten desgranar el model. Com a inconvenients, s'ha de destacar la inestabilitat de l'algorisme amb conjunts d'entrenament petits (pocs individus per branca) i que el criteri de guany d'informació tendeix a afavorir a atributs amb més possibles valors. En aquest conjunt de dades particulars, a més, l'algorisme

ha donat com a resultat un error bastant elevat, en comparació a la resta d'algorismes avaluats. El seu temps d'execució ha sigut similar al del kNN amb k=7 (0.192s).

- L'algorisme Naïve Bayes ha demostrat un bon rendiment computacional (0.172s) i ha retornat, per a aquest exemple, resultats similars al kNN amb k=7. El seu principal inconvenient és la seva inestabilitat en entrenaments amb pocs individus (les classes sense representació tenen una probabilitat d'ocurrença 0). Aquest inconvenient, però, es pot resoldre parcialment amb tècniques de suavitzat (probabilitat estàndar per a individus sense representació en el conjunt d'entrenament). Destaca la baixa expressivitat del mètode, que no admet massa paràmetres d'ajust.
- Per últim, el SVM ha presentat, en les seves variants lineal i radial, el millor resultat entre els algorismes utilitzats (no així el polinòmic). Notem que, com a pas previ a l'aplicació d'aquest algorisme, valdria la pena analitzar la forma de les dades amb alguna tècnica de reducció de la dimensionalitat com el PCA o el MDS, per tal de saber quin tipus de kernel ajustaria millor les dades. Una de les avantatges del SVM és que permet construir un model, la qual cosa millorarà el rendiment en aplicacions on-line. D'altra banda, la gran quantitat de paràmetres existents en el model permet l'ajust per tal d'optimitzar el rendiment de l'algorisme.

Com a comentari global, val la pena mencionar la gran utilitat trobada a la llibreria scikit-learn, que ens ha permet resoldre la major part de tasques de la pràctica (des de la normalització fins als classificadors) amb poques línies de codi i d'una forma fàcilment assimilable i comprensible per a tercers, gràcies a l'excel·lent documentació disponible.

Caben millores en la implementació resolta en la pràctica, com un anàlisi de components principals previ que ens permeti recollir la forma de les dades en un pla 2d per tal d'optimitzar els paràmetres d'alguns algorismes com l'SVM o una classificació en 2 passos que aprofiti el guany d'informació que aporta saber si la primera classe està ben classificada: es podria construir un classificador en 2 fases que, una vegada sàpigues el canal (amb una probabilitat d'encert elevada), aprofités aquesta informació per a predir la regió. Aquesta aproximació presentaria, com a dificultat addicional, el fet d'haver de treballar amb variables nominals i numèriques conjuntament.