Marc Garcia i Mullon 158637
Aurel Ioan Patrutiu 158689
Universitat Pompeu Fabra - 17/03/2017

Explicació del Codi

El codi està dividit en tres fitxers:

- main.py:

  Mira el nombre de paraules que l'usuari vol tenir com a paraules més comunes i crea els obejctes “Most_common” i “OutputWeka” en aquest ordre. També trobem el directori on es troben els fitxers que haurem de llegir.

- most_common.py:

  Aquí tenim la funció “readAndMake()” que donant-li el nombre de paraules que l'usuari vol tenir com a paraules més comunes i el directori on es troben els fitxers que haurem de llegir, et calcula les paraules més comunes en tots els fitxers. Per fer-ho el que fem és utilitzar la llibreria de Python nltk que ens permet separar per paraules el text. Un cop hem separat per paraules el text mirem si la paraula ja està com a “key” al nostre diccionari, si hi és, sumem 1 al valor del diccionari d’aquella “key”. Si no es troba al diccionari, el que fem és crear aquella “key” i inicialitzar el seu valor a 1. Al codi per no haver d'iterar tant pel diccionari, hem utilitzat una llibreria de Python anomenada collections que el que ens permet és contar quantes repeticions d’aquella paraula hi ha, i ho inserim directament. Aquest diccionari el guardem dins una variables anomenada “counts” que esta dins el objecte “Most_common”.

- outputWeka.py:

  Aquí tenim la funció “readAndMake()” que ens crear el fitxer arff amb el tant per cent de vegades que surt cada una de les paraules del vector que ens ha creat el most_common. I això ho fa per cada un dels fitxers.
  Per fer-ho fem servir el mateix procediment que amb la funció “readAndMake” de la classe “Most_common”.

Marc Garcia i Mullon 158637
Aurel Ioan Patrutiu 158689
Universitat Pompeu Fabra - 17/03/2017

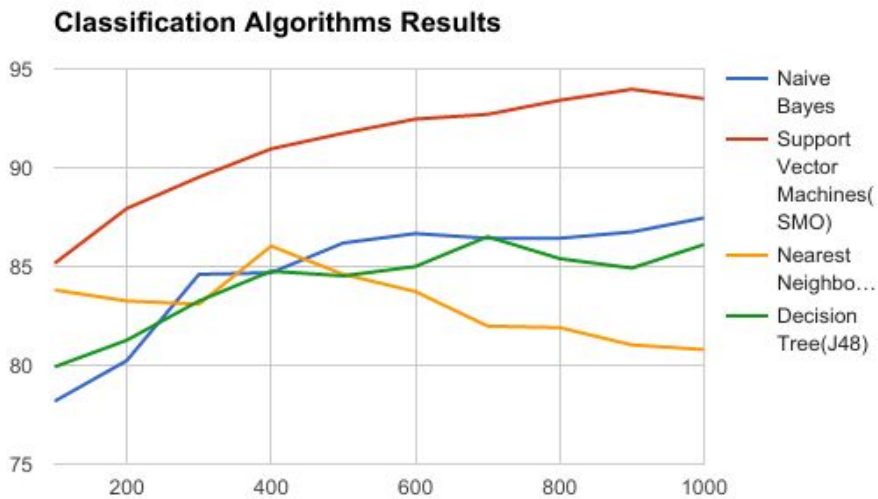
WEKA

Una vegada executat el nostre programa i obtinguts els fitxers (.arff) amb els documents vectoritzats introduïm aquests fitxers al programa WEKA per tal d'executar els diferents algoritmes de classificació. De tots els algoritmes que Weka ens ofereix hem escollit els següents quatre.

- Naive Bayes : Classificador probabilístic molt simple però amb resultats fiables.
- Support Vector Machines (SMO): Classificador supervisat basat a separar les classes mitjançant un hiperplà equidistant a les instàncies més properes.
- KNN: Classificador supervisat basat a classificar les instàncies en funció als "k" veïns més propers a la nova instància.
- Decisió Tree (J48): Classificador supervisat basat en un arbre generat a partir de petites regles.

Una vegada executats tots aquest algoritme sobre tots els nostres fitxers .arff hem obtingut per cada fitxer els resultats afegits als Annexos d'aquest document.

Per tal d'analitzar les dades obtingudes hem col·locat tots els resultats en la següent gràfica:



Com podem veure, en la imatge anterior tenim els resultats en % de les execucions amb les diferents quantitats de paraules. És a dir, hem generat un fitxer per cada execució amb 100, 200, 300 i així fins a 1000 característiques al vector de cada document analitzat. Una vegada introduïdes les dades a Weka hem executat els quatre algoritmes sobre aquests fitxers. Analitzant les dades ens podem adonar que com més característiques té els nostres fitxers de resultats més accuracy tenen els nostres classificadors. Com més informació

tenen els nostres classificadors per poder entrenar l'algoritme, millor encertarà les classes de les noves instàncies de test.
Dit això i mirant més les execucions veiem que dels quatre algoritmes el que millors resultats ens dóna es Suport Vector Machines. Això és perquè el core d'aquest algoritme és molt sòlid i fa una molt bona separació de classes. Aquest comença amb una acuracy del voltant de 80% i en quant anem augmentant el nombre de característiques veiem que va pujant de mica en mica fins al 95%.
La resta de classificadors també van pujant la seva acuracy a l'hora de classificar les instàncies de test però amb molta menys precisió que SMO.

Per tant com a conclusió de les execucions anteriors extraiem que com més característiques introduïm al vector que representa un document, millor funcionaran els nostres classificadors. Això no serà sempre d'aquesta manera, ja que des de cert punt sabem que els algoritmes començaran a estar sobre entrenat i per tant hi haurà overfiting. Malgrat això amb els números que hem provat hem obtingut molt bons resultats.

Marc Garcia i Mullon 158637
Aurel Ioan Patrutiu 158689
Universitat Pompeu Fabra - 17/03/2017

Annexos

Output_100.arff	
Algorithm	**Accuracy**
Naive Bayes	78,1764%
Super Vector Machines(SMO)	85,1587%
Nearest Neighbourhood(iBK)	83,8095%
Decision Tree(J48)	79,9206%

Output_200.arff	
Algorithm	**Accuracy**
Naive Bayes	80.2381 %
Super Vector Machines(SMO)	87.9365 %
Nearest Neighbourhood(iBK)	83.254 %
Decision Tree(J48)	81.2698 %

Output_300.arff	
Algorithm	**Accuracy**
Naive Bayes	84.6032 %
Super Vector Machines(SMO)	89.5238 %
Nearest Neighbourhood(iBK)	83.0952 %
Decision Tree(J48)	83.254 %

Output_400.arff	
Algorithm	**Accuracy**
Naive Bayes	84.6825 %
Super Vector Machines(SMO)	90.9524 %
Nearest Neighbourhood(iBK)	86.0317 %
Decision Tree(J48)	84.7619 %

Marc Garcia i Mullon 158637
Aurel Ioan Patrutiu 158689
Universitat Pompeu Fabra - 17/03/2017

Output_500.arff	
Algorithm	**Accuracy**
Naive Bayes	86.1905 %
Super Vector Machines(SMO)	91.746 %
Nearest Neighbourhood(iBK)	84.6032 %
Decision Tree(J48)	84.5238 %

Output_600.arff	
Algorithm	**Accuracy**
Naive Bayes	86.6667 %
Super Vector Machines(SMO)	92.4603 %
Nearest Neighbourhood(iBK)	83.7302 %
Decision Tree(J48)	85%

Output_700.arff	
Algorithm	**Accuracy**
Naive Bayes	86.4286 %
Super Vector Machines(SMO)	92.6984 %
Nearest Neighbourhood(iBK)	81.9841 %
Decision Tree(J48)	86.5079 %

Output_800.arff	
Algorithm	**Accuracy**
Naive Bayes	86.4286 %
Super Vector Machines(SMO)	93.4127 %
Nearest Neighbourhood(iBK)	81.9048 %
Decision Tree(J48)	85.3968 %

Marc Garcia i Mullon 158637
Aurel Ioan Patrutiu 158689
Universitat Pompeu Fabra - 17/03/2017

Output_900.arff	
Algorithm	**Accuracy**
Naive Bayes	86.746 %
Super Vector Machines(SMO)	93.9683 %
Nearest Neighbourhood(iBK)	81.0317 %
Decision Tree(J48)	84.9206 %

Output_1000.arff	
Algorithm	**Accuracy**
Naive Bayes	87.4603 %
Super Vector Machines(SMO)	93.4921 %
Nearest Neighbourhood(iBK)	80.7937 %
Decision Tree(J48)	86.1111 %

Output_1100.arff	
Algorithm	**Accuracy**
Naive Bayes	87.619 %
Super Vector Machines(SMO)	93.8889 %
Nearest Neighbourhood(iBK)	79.8413 %
Decision Tree(J48)	86.6667 %