PROYECTO ESTÁNDAR

INTRODUCCIÓN AL RECONOCIMIENTO DE PATRONES 2015

CLASIFICACIÓN DE BOSQUES POR INFORMACIÓN CARTOGRÁFICA

Integrantes:

José Ignacio Reyes
Facundo Artagaveytia

Tutores:

Pablo Cancela
Sergio Martínez

Contenido

RESUMEN:

El presente documento fue realizado en el marco del proyecto de fin de curso de la asignatura 'Reconocimiento de Patrones' de la Universidad de la República (UdelaR), en el cual se pretende se utilicen los métodos, técnicas y algoritmos de clasificación estudiados durante el curso, así como incursionar en otros nuevos.

El conjunto de datos a clasificar consta de 7612 bosques nativos del estado de Colorado, Estados Unidos, donde a partir de los valores que toman 54 características diferentes, se desea entrenar un clasificador lo más efectivo posible. Se desea alcanzar un desempeño de al menos 82% al validar el clasificador entrenado sobre un conjunto de test que fue proporcionado sobre el final del proyecto.

En un principio se utilizaron algunos algoritmos clásicos de clasificación, para luego de aplicar algunas técnicas, verificar la influencia de las mismas en el desempeño del clasificador. Los algoritmos utilizados en un principio fueron el *NaiveBayes*, el *J48* y el *K-NN*. El entorno utilizado para hacer estas pruebas iniciales fue el *Weka*.

Tras aplicar los algoritmos mencionados, el desempeño obtenido no superaba el 78% en ninguno de los casos. A continuación se trabajó sobre los *outliers*, realizando un programa en *Matlab*, tras verificar que existían datos que convenía no fueran tenidos en cuenta a la hora de entrenar el clasificador, y se evaluó la conveniencia de la eliminación de dichos datos. También se analizaron las características de tipo binario, sobre las cuales se aplicaron algunas modificaciones, también utilizando la herramienta *Matlab* para agregar información al clasificador y reducir la dimensionalidad del problema.

Finalmente se decidió probar con otros algoritmos del tipo 'árbol de decisión', dado el buen resultado en 'crudo' del J48 y fue así como se dio con el algoritmo *RandomForest*, con el cual, luego de aplicadas las técnicas mencionadas, se obtuvo un desempeño por encima de 84%.

OBJETIVO:

Se desea clasificar el tipo mayoritario de árboles en bosques nativos ubicados en el norte del estado de Colorado, en EEUU. Se debe diseñar un sistema de reconocimiento de patrones que permita clasificar el tipo de árboles. El sistema debe maximizar el porcentaje de acierto asumiendo que las clases de tipos de bosque son equiprobables. Como objetivo del presente proyecto se solicita cumplir con los siguientes cuatro puntos.

En primer lugar, se pide que el sistema alcance un desempeño de 82% evaluado con el conjunto de entrenamiento disponible. Se debe detallar la metodología seguida para mostrar que se cumple dicho desempeño.

En segundo lugar se pide explorar alguna forma de mejorar la información de tipo de suelo, agregando y/o sustituyendo las características existentes por otras. Se debe evaluar si esto tiene algún impacto en el desempeño del sistema de clasificación.

En tercer lugar se pide explorar el impacto que tiene aplicar alguna técnica de edición sobre los datos.

Por último, se debe estimar el desempeño esperado para el conjunto de test, desconocido a priori.

ENFOQUE:

Se consideraron las características Elevation (Elevación en metros), Aspect (Azimut del terreno en grados), Slope (pendiente del terreno en grados), Horizontal_Distance_To_Hydrology (Distancia horizontal a la fuente de agua superficial más cercana), Vertical_Distance_To_Hydrology (Distancia vertical a la fuente de agua superficial más cercana), Horizontal_Distance_To_Roadways (Distancia horizontal a la ruta más cercana), Hillshade_9am (Índice de sombra a las 9am, en el solsticio de verano), Hillshade_Noon (Índice de sombra al mediodía, en el solsticio de verano), Hillshade_3pm (Índice de sombra a las 3 pm, en el solsticio de verano), Horizontal_Distance_To_Fire_Points (Distancia horizontal a puntos donde hubo comienzo de incendios), Wilderness_Area (Nombre del área en la que está situado), Soil_Type (Tipo de suelo) y Cover_Type (Tipo de bosque, que es la clase a estimar).

A partir de las características mencionadas, se aplicaron diversas técnicas y algoritmos utilizados durante el curso para alcanzar el desempeño objetivo de 82%.

En este trabajo no se pretendió profundizar en alguna técnica particular, sino intentar cumplir los objetivos planteados analizando principalmente el conjunto de estrategias vistas a lo largo del curso.

DATOS:

En primer lugar se trabajó sobre el conjunto de datos de entrenamiento brindado por los docentes del curso, a partir del cual se intentaron cumplir los cuatro objetivos planteados. Posteriormente se evaluó el clasificador resultante sobre el conjunto de test para verificar si la estimación realizada fue correcta.

En un principio se contaba solamente con el conjunto de entrenamiento, y sobre el conjunto de test se sabía únicamente que sería de las mismas características que el anterior, y que tendría igual cantidad de patrones por clase.

SOFTWARE:

Para la realización del presente proyecto se utilizó el software *Weka* (*Waikato Environment for Knowledge Analysis*, en español «entorno para análisis del conocimiento de la Universidad de *Waikato*»). El mismo es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de *Waikato*. El paquete *Weka* contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades.

También se utilizó la herramienta *Matlab*, para procesar los datos.

ETAPA 1

Introducción

En esta primera etapa se contaba solamente con el conjunto de datos de entrenamiento, el cual está compuesto por 7612 instancias, repartidas de forma no equitativa entre las 7 clases posibles. A partir de dicho conjunto de datos se intentó entrenar diversos clasificadores utilizando distintos algoritmos, con el fin de obtener el mayor desempeño posible.

El conjunto de test sobre el cual se deberá probar el desempeño de los clasificadores en una etapa posterior tendrá las mismas características que los datos de entrenamiento y tendrá una cantidad igual de elementos por cada clase.

Resample y evaluación inicial de algoritmos:

En primer lugar se cargó en *Weka* el archivo ***forest_train_set.csv***, conteniendo los datos de entrenamiento. La primer acción a realizar fue la de aplicar el filtro *filter.unsupervised.attribute.NumericToNominal,* al atributo de clase *Cover_type*, esto se realizó dado que el tipo de clase fue cargado desde el archivo *csv* como un atributo numérico, cuando en realidad es nominal, dado que no puede tomar valores intermedios entre las diferentes clases posibles. De no haber aplicado este filtro, no sería posible emplear diversos algoritmos, que requieren que la clase sea un atributo nominal, como lo son el *j48* y el *naiveBayes*. Por razones similares se aplicó también el filtro *filter.unsupervised.attribute.NumericToBinary*, sobre las características binarias las cuales habían sido cargadas como numéricas. El nuevo conjunto de datos se almacenó como ***forest_train_set.arff***

Motivados por el hecho de que los datos de test tienen una cantidad igual de elementos por cada clase, y que las clases son equiprobables, es que se decidió aplicar un filtro sobre el conjunto original para equilibrar la cantidad de patrones de cada clase. Una forma de realizar esto, es aplicando el filtro *filter.supervised.instance.Resample*, con el parámetro *biasToUniformClass* seteado en 1. El valor de *filter.supervised.instance.sampleSizePorcent*, el cual determina qué porcentaje de los datos originales se mantiene, se fija de forma tal que todas las clases tengan la misma cantidad de elementos que la clase con menor cantidad de instancias, de forma de no generar patrones inexistentes, repitiendo algunos de la clase correspondiente. Para nuestro caso, el porcentaje para el cual se da dicha situación es 83.7%, a partir del cual se genera un conjunto de datos con 910 instancias por clase. Además se fijó el parámetro *filter.supervised.instance.noReplacement* en true, de modo que a la hora de hacer el *Resample* no se vuelvan a seleccionar patrones que ya hayan sido seleccionados. Esto último es de vital importancia en esta etapa, dado que en un principio, vamos a evaluar el desempeño del clasificador sobre el mismo conjunto de entrenamiento. Si este tuviese valores duplicados, el desempeño del clasificador mejoraría considerablemente, para el caso puntual de usar el conjunto de

entrenamiento como conjunto de test, cosa que no queremos, dado que intentamos estimar cual va a ser el desempeño del clasificador cuando lo apliquemos a otro conjunto diferente.

Previo a aplicar el *resample*, se consideró el hecho de que a pesar de que el conjunto de datos resultante va a tener igual peso en todas las clases al igual que el conjunto de test, lo cual es favorable, también estaríamos incurriendo en una pérdida de datos, lo cual no es del todo favorable. Tomando en cuenta que el conjunto de datos de entrenamiento tiene un total de 7612 instancias, y que el *resample* generaría un nuevo conjunto que contiene 910 instancias de cada clase, es decir 6370 patrones, estaríamos resignando poco más del 16% de los datos iniciales.

Para evaluar cuál de los dos argumentos tiene mayor peso se decidió dividir el conjunto de entrenamiento en dos partes. El programa ***división_conjunto_train.m*** para *Matlab* realiza dicha tarea, devolviendo los archivos ***forest_train_set _div1.csv*** y ***forest_train_set _div2.csv***. El primero de los dos conjuntos será utilizado como conjunto de entrenamiento, mientras que el segundo como conjunto de test, para testear la eficacia de aplicar el *resample* sobre los datos. Para ello se, cargó en weka el archivo ***forest_train_set _div2.csv***, al cual se le aplicaron los filtros, *NumericToNominal* y *NumericToBinary*, tal como se había hecho anteriormente, y se aplicó un *resample* de modo de equilibrar las clases. El conjunto resultante se guardó como ***forest_train_set _div2.arff*** y será tomado como el conjunto de test, el *resample* fue aplicado de modo que el mismo tenga igual peso en todas las clases como el conjunto de test que será proporcionado más adelante.

A continuación se cargó en weka en archivo ***forest_train_set _div1.csv*** y se le aplicaron los mismos filtros sobre las características, el archivo resultante se guardó como ***forest_train_set _div1.arff***. A continuación se realizó lo siguiente, en primer lugar se evaluó el rendimiento de algunos algoritmos básicos utilizados durante el curso sobre el conjunto ***forest_train_set _div2.arff*** y luego se aplicó un *resample* para equilibrar las clases y se volvió a evaluar los mismos algoritmos.

Previo a aplicar el *resample* se aplicaron los algoritmos *NaiveBayes* (con estimador kernel), *J48* y *K-NN* (regla del vecino más cercano con *crossValidate* para estimar el k óptimo) sobre el conjunto de datos obteniendo los siguientes desempeños:

Sin Resample	***NaiveBayes***	***J48***	***K-NN(1)***
forest_train_set _div1.arff	68.41%	73.6925%	75.3219%

Los desempeños anteriores son el promedio de los desempeños obtenidos tras correr los algoritmos con 10 semillas diferentes. Este será el método utilizado en el correr del presente trabajo para evaluar los algoritmos. La varianza de los desempeños fue

siempre inferior a 0.06. De ahora en más no se hará hincapié en la varianza siempre y cuando esta se mantenga por debajo de 0.1.

Luego de aplicar el *resample* sobre el conjunto de entrenamiento ***forest_train_set _div1.arff*** se aplicaron los mismos algoritmos obteniendo los siguientes resultados:

Con Resample	***NaiveBayes***	***J48***	***K-NN(1)***
forest_train_set _div1.arff	65.5453%	71.4586%	72.4047%

Como podemos apreciar, tras aplicar el *resample* todos los clasificadores se equivocaron en mayor medida, que sin aplicarlo. Esto indica que el hecho de perder datos, nos perjudica más de lo que nos puede llegar a beneficiar el hecho de entrenar con un conjunto de datos que tenga el mismo peso en las clases que el conjunto de test. Vale aclarar también que en este caso, a pesar de que no están del todo equilibradas, las clases tienen un número de instancias relativamente similar, por lo cual decidimos no aplicar el filtro *resample* sobre los datos de entrenamiento.

A continuación se decidió aplicar algunos algoritmos de clasificación sobre el conjunto de entrenamiento ***forest_train_set.arff***, para obtener algunos desempeños iniciales contra los cuales comparar luego de aplicadas algunas técnicas. Para la evaluación de los clasificadores se utilizó *cross-validation* en 10 grupos. Esto significa que se divide el conjunto de entrenamiento en 10 grupos. Cada grupo será utilizado para evaluar el desempeño de un clasificador entrenado con los patrones de los otros 9 grupos. Posteriormente se promedian los desempeños obtenidos, siendo el resultado el desempeño del clasificador. Se aplicaron los algoritmos *NaiveBayes* (con estimador kernel), *J48* y *K-NN* (regla del vecino más cercano con *crossValidate* para estimar el k óptimo) sobre el conjunto de datos obteniendo los siguientes desempeños.

Cross-validation	***NaiveBayes***	***J48***	***K-NN(1)***
forest_train_set_1.arff	69.351%	77.0888%	78.8492%

Como se puede observar el algoritmo *K-NN* es el que obtiene el mejor desempeño, seguido por el *J48*.

Tras aplicar los filtros y utilizar los métodos de validación mencionados, podemos afirmar que el resultado debería aproximarse al que se obtendrá cuando se valide el clasificador con el conjunto de test. Por lo tanto estamos aún lejos del 82% objetivo.

Edición de datos - Outliers:

Luego de analizar los datos con profundidad, se constató que algunos patrones tomaban valores extremos en algunas características, que no mostraban demasiada coherencia con el resto de los patrones de la misma clase. Por ejemplo, algunos de los bosques del conjunto de entrenamiento, tomaban para la característica *Elevation*,

valores entre 0 y 1000 metros, lo cual intuimos debe ser un error en los datos, dado que el territorio del estado de Colorado se encuentra por encima de los 1000 metros en toda su extensión.

Por lo tanto, el siguiente paso que se decidió tomar fue el de revisar los datos para analizar los patrones que puedan ser considerados como *outliers.* Remover los outliers consiste en eliminar de la base de datos las muestras irrelevantes o redundantes que no favorezcan al aprendizaje, de esta manera se reduce el coste computacional y se aumenta la precisión del modelo, ya que se han eliminado los elementos que perjudican el aprendizaje.

Para ello se creó en Matlab el programa ***sacar_outliers.m***. Dado que en la mayoría de los casos, es solamente una de las características del patrón, la que toma un valor que no condice con los restantes patrones de la clase, en lugar de eliminar dichos patrones, se decidió sustituir los valores extremos por valores más usuales para la clase en cuestión. En conclusión, lo que hace el programa es sustituir las características que se encuentran fuera de un rango determinado, por un valor dentro del rango. Dicho rango fue determinado tras inspección visual de los valores que toma cada característica, para los patrones de cada clase. Primero se ordenaron dichos valores de menor a mayor, y luego se determinaron los limites superior e inferior a partir de los cuales es notable que se trata de casos excepcionales. En algunos casos era notoria la presencia de dichos valores extremos (Figura 1), mientras que en otros casos se decidió mantener todos los patrones tal cual estaban (Figura 2).

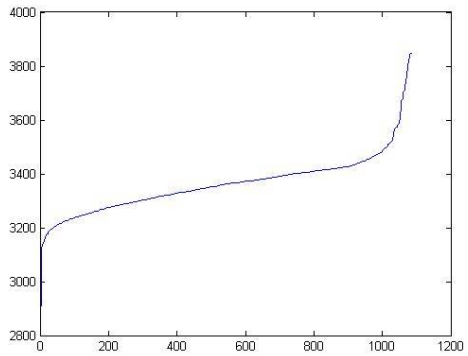


Figura 1

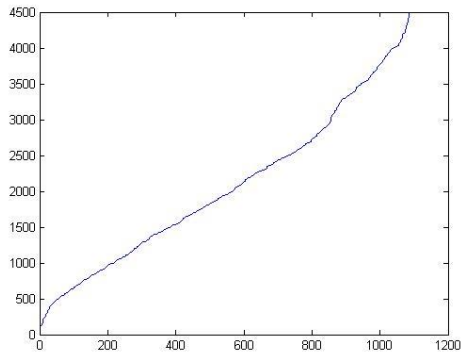


Figura 2

A la hora de quitar los *outliers*, entramos en la misma disyuntiva que en el caso del filtro *Resample*, dado que no sabemos hasta donde, quitar los valores extremos favorece nuestro clasificador, debido a que se entrena tomando valores más certeros de cada clase, o lo perjudica, dado que va a ser menos probable que clasifique dichos valores extremos de forma correcta, si estos se presentan en el conjunto de test.

Para resolver dicho dilema volvimos sobre los conjuntos obtenidos a partir de dividir el conjunto de entrenamiento en dos. En este caso se evaluó el desempeño de los

mismos tres clasificadores para el caso de sacar los outliers del conjunto de entrenamiento ficticio ***forest_train_set_div1.csv***. Tras correr el programa ***sacar_outliers.m*** sobre el archivo mencionado, se obtuvo el nuevo conjunto ***forest_train_set_div1_sinoutliers.csv***, el cual, tras cargarlo en weka y aplicarle los filtros *NumericToNominal* y *NumericToBinary* a las características correspondientes fue almacenado como ***forest_train_set_div1_sinoutliers.arff.***

Se aplicaron los algoritmos *NaiveBayes* (con estimador kernel), *J48* y *K-NN* (regla del vecino más cercano con *crossValidate* para estimar el k óptimo) sobre el conjunto de datos obteniendo los siguientes desempeños al evaluarlo sobre el conjunto de test ***forest_train_set_div2.arff***.

Sin outliers_1	***NaiveBayes***	***J48***	***K-NN(1)***
forest_train_set_div1_sinoutliers.arff	67.1748 %	72.5624 %	71.8791 %

Como pudimos apreciar, los desempeños disminuyeron en gran medida luego de aplicar la remoción de los *outliers*. Previo a descartar la aplicación de dicha técnica se decidió ampliar los límites establecidos, y volver a evaluar los clasificadores. El programa ***sacar_outliers_2.m*** quita los outliers al igual que ***sacar_outliers.m***, pero tomando un rango de valores "correctos" superior para cada característica. Los valores obtenidos tras aplicar los mismo algoritmos al conjunto de datos devuelto por ***sacar_outliers_2.m*** fueron los siguientes.

Cross-validation	***NaiveBayes***	***J48***	***K-NN(1)***
forest_train_set_div1_sinoutliers.arff	68.083 %	70.5125 %	72.7727 %

Claramente los desempeños volvieron a disminuir, por lo cual concluimos que sacar los *outliers* perjudica el entrenamiento de nuestro clasificador.

Algoritmos alternativos

A continuación se decidió utilizar el método de las máquinas de vectores de soporte para evaluar el desempeño del algoritmo. Utilizando el algoritmo SMO, con un kernel RBF, fijando la constante de penalización 'c' en 100 y gamma en 0.1, se obtuvo un desempeño de 76.7604% tras aplicarlo sobre el archivo ***forest_train_set.arff***, validando con *cross-validation* en 10 grupos. Este es un desempeño bastante interesante para tratarse de un primer intento a "ciegas", más adelante intentaremos optimizar los parámetros del algoritmo para ver hasta dónde podemos llegar con este método.

Dado que con el J48 se obtuvo un resultado bastante bueno, con los datos prácticamente en crudo, se decidió intentar con otros algoritmos del tipo árbol de decisión. El algoritmo *RandomForest*, es uno de los algoritmos más efectivos que

existen para clasificar datos. Su funcionamiento se basa en dos principios, el *bagging* sobre los datos proporcionados a cada árbol y la aleatoriedad en la característica que define la división de los datos en cada nodo. Bagging es el acrónimo de bootstrap aggregation. En general cuando se promedian varios modelos se obtiene un mejor ajuste que cuando se utiliza un solo modelo de algoritmo de aprendizaje. La idea básica es remuestrear los datos y calcular las predicciones sobre el conjunto de datos remuestreados. Al promediar varios modelos conjuntamente se obtiene un mejor ajuste debido a que se mitigan tanto los modelos con sesgo como los modelos con alta varianza.

Aplicando dicho algoritmo sobre los datos ***forest_train_set.arff*** con validación mediante *cross-validation*, se obtuvo un desempeño de 83.6311 %, lo cual supera el umbral solicitado.

Optimización de parámetros

Primeramente, dado el alto desempeño obtenido con un algoritmo del tipo 'árbol de decisión' (*RandomForest*) se optó por intentar perfeccionar el J48 en lo que refiere a la elección de sus parámetros. Para ello se utilizó el meta clasificador *CVParameterSelection,* en el cual se evaluó el parámetro C (factor de confianza para la poda) desde 0.2 hasta 0.45, en pasos de a 0.05. Finalmente se obtuvo que el C óptimo es el de 0.25 que se había utilizado, por lo cual no se puede mejorar el desempeño de esta forma.

A continuación se trabajó sobre los parámetros del algoritmo SMO. Tras evaluar los valores óptimos de 'c' y gamma para SMO con kernel RBF de 0 a 200 y -5 a 5 respectivamente, utilizando *GridSearch*, se obtuvieron los valores c = 60 y gamma 0.01, para los cuales el clasificador alcanza un desempeño de 77.4411 %.

En cuanto al algoritmo *RandomForest* con el cual se había obtenido un desempeño máximo de 83.6311 %, tras variar los parámetros *numFeatures* y *numTrees*, seteando los mismos en 10 y 300 respectivamente, se alcanzó un desempeño de 84.3405%.

Características binarias:

Analizando las características individualmente, se puede observar que hay dos conjuntos de características binarias que pueden ser agrupadas en solamente dos características, y de encontrarse algún motivo que permita ordenarlas, se puede agregar así información al conjunto de datos. Además realizando esto, se estaría reduciendo la dimensionalidad del problema considerablemente, lo cual es también un aspecto positivo.

En primer lugar se tomaron las características *Wilderness_Area* y se codificaron dándoles valores del 1 al 4. Las áreas se ordenaron según su ubicación geográfica de oeste a este.

Luego se ordenaron las características Soil_Type, asignándole a cada uno de los tipos de suelos un número del 1 al 40. En una primera aproximación se utilizó el orden que traen por defecto, detallados en el anexo de la descripción del problema.

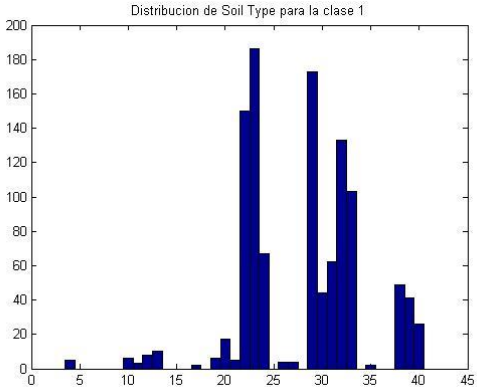
Para llevar a cabo la creación de estas nuevas características y la asignación de los valores correspondientes a cada patrón, se escribió un programa en Matlab llamado ***binarias1.m***, el cual toma el archivo ***forest_train_set.csv*** con las características preprocesadas y retorna en el archivo ***forest_train_set_binarias1.csv*** las nuevas características según lo descrito anteriormente.

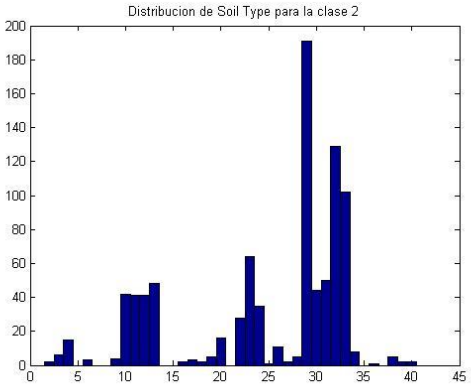
De igual forma que al principio, se aplicó el filtro *filter.unsupervised.attribute.NumericToNominal* al atributo de clase *Cover_type*. En este caso también se aplicó el mismo a las nuevas características obtenidas a partir de las características binarias, dado que son tomadas como numéricas, pero estas nunca pueden llevar un valor que no sea un número entero, en el rango que le corresponda (1 a 4 y 1 a 40).

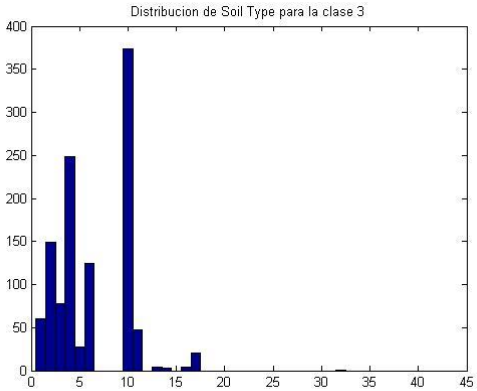
Tras aplicar el algoritmo que hasta el momento había dado mejores resultados (Random_Forest) se obtuvo un desempeño promedio de 83.9776 %.

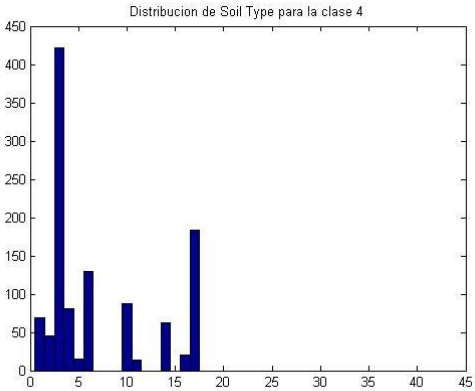
Este valor es apenas inferior al obtenido sin este agrupamiento de clases binarias, pero se cuenta con la clara ventaja de haber reducido la dimensionalidad considerablemente.

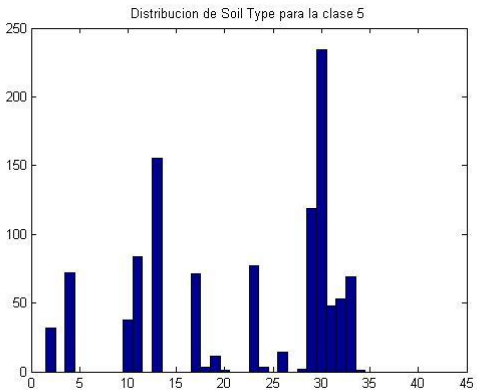
Luego se intentó mejorar el desempeño del clasificador observando las características binarias para observar alguna particularidad que permitiera generar una nueva característica que le brindara más información al clasificador. Por esto se graficaron los histogramas de las 7 clases para la característica Soil_Type.

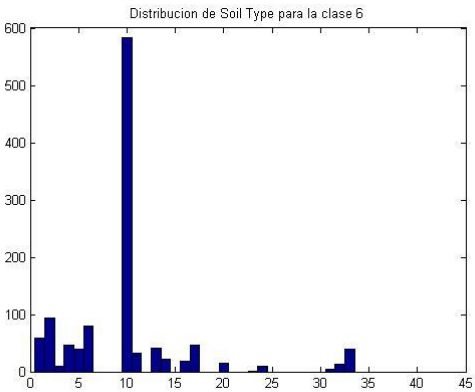


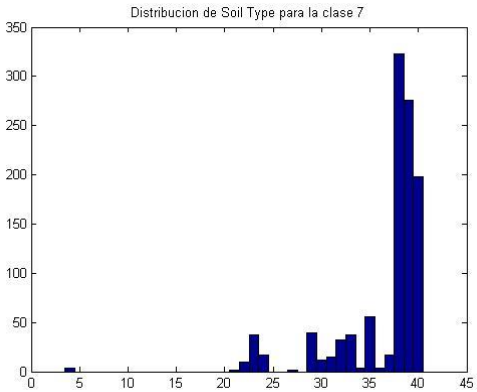












En una primera instancia se buscó mejorar la eficiencia reordenando los tipos de suelo según la rocosidad de los mismos en lugar de dejar el orden que se tenía por defecto.

Con esta nueva forma de ordenar las características Soil_Type se obtiene un peor desempeño con respecto al orden que tenían por defecto (83.2107%)**.**

Por último se crea una nueva característica que surge de observar los histogramas anteriores. Esta les asigna cero a los patrones cuyo valor de Soil_Type es menor o igual a 17, y 1 en caso contrario. Esta idea nace al ver que varias clases tienen patrones o bien con Cover_Type menores a 17 o con mayores a 17.

Luego de juntar todas las binarias en 2 caracteristicas del 1 al 4 y del 1 al 40, y además crear la nueva característica descrita anteriormente se obtiene un rendimiento del clasificador de 83. 8932 %.

Siguiendo el mismo razonamiento anterior se intentó crear nuevas clases para mejorar el desempeño. Estas nuevas características agrupan de diferentes formas diferentes tipos de suelos a partir de los histogramas de cada clase. Todas las opciones que se probaron redujeron el desempeño del clasificador.

ETAPA 2

Introducción

En esta segunda etapa se cuenta con el conjunto de *test* proporcionado por los docentes del curso y a partir del mismo se validará el mejor clasificador obtenido al momento.

Validación

Dado que en la etapa 1 concluimos que, tanto realizar el resample de datos, como sacar los outliers, reduce el desempeño de nuestro clasificador, y además el trabajo realizado sobre las características binarias, no logro incrementar el mismo, aunque si disminuir la dimensionalidad, el mejor clasificador obtenido al momento sería aplicar el algoritmo *RandomForest* con los atributos numFeatures y numTrees seteados en 10 y 300 respectivamente. Tras aplicar dicho algoritmo sobre el conjunto de entrenamiento el desempeño obtenido fue de 84.3405%, en las condiciones que creemos son más favorables para estimar el desempeño sobre el conjunto de test.

Finalmente, tras aplicar el clasificador mencionado sobre el conjunto de test, el desempeño resultante fue de 84.4233%

CONCLUSIÓN

A modo de conclusión podemos decir que se cumplió con los objetivos del proyecto, dado que se exploraron métodos de edición de los datos, trabajando sobre los *outliers* y sobre el peso de las clases. Además se exploraron diversas formas de modificar las características binarias para reducir tanto la dimensionalidad como agregar información al sistema. El objetivo de estimar el desempeño sobre el conjunto de test proporcionado al final del proyecto también fue cumplido satisfactoriamente dado que se esperaba un desempeño de 84.3405% y se obtuvo uno de 84.4233%.

Algunos puntos que consideramos vale la pena resaltar fue el análisis realizado respecto a la aplicación de los métodos de edición de datos, concluyendo que no favorecían el desempeño del clasificador, lo cual fue verificado posteriormente con el conjunto de test.

Como aspectos negativos del proyecto podríamos marcar el hecho de que no se logró añadir información útil al sistema a partir del análisis de las características.

Como trabajo futuro se plantea el análisis de las características 1 y 2, y 3 y 6, clases entre las cuales más se confunde el clasificador obtenido. Donde una primera acción a tomar sería aplicarle otro clasificador (generado solo con los datos de entrenamiento de las clases que se confunden) a las clases que fueron clasificadas como 1 y 2, o 3 y 6.