Introducción al Reconocimiento de Patrones

Facultad de Ingeniería - UDELAR

Proyecto Estándar:

Clasificación de Bosques por Información Cartográfica

Sebastián Bugna
Federico Nin

Tutores: Alicia Fernández, Sergio Martinez

Diciembre
2015

Índice

1. Descripción del Problema

1.1. Introducción

El problema consiste en clasificar el tipo mayoritario de árboles en bosques nativos. El estudio incluye cuatro áreas de bosques naturales, situados en el norte del estado de Colorado en EEUU, como se muestra en la Fig.1. Cada observación corresponde a un área de 30x30 metros. Se debe predecir el tipo de árboles de cobertura que están presentes. Los 7 tipos de árboles son:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

El conjunto de entrenamiento tiene 7560 observaciones con 54 características relevadas y está etiquetado con los 7 tipos de cobertura.

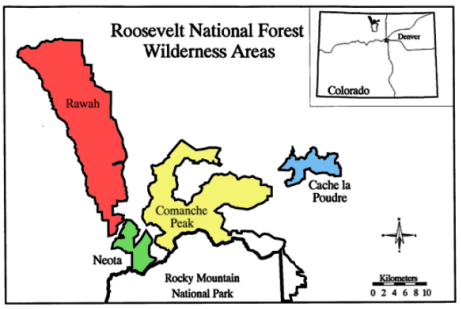


Figura 1: WildernessAreas Parque Nacional Roosvelt

1.2. Objetivos

Se desea diseñar un sistema de reconocimiento de patrones que permita clasificar el tipo de árboles o cobertura presente a partir de muestras sin etiquetar. El sistema debe maximizar el porcentaje de acierto asumiendo que las clases de tipos de bosque son equiprobables.

Se trabajará en primera instancia con el conjunto de datos de entrenamiento. Se hará una posterior evaluación con un conjunto diferente de test.

Se debe cumplir con los siguientes puntos:

- El sistema debe tener un porcentaje de acierto de al menos el 82 % evaluado con el conjunto de datos de entrenamiento disponible.

- Explorar una forma de mejorar la información del tipo de suelo, agregando y/o sustituyendo las características existentes por otras. Evaluar si esto tiene impacto en el desempeño.

- Explorar el impacto que tiene aplicar alguna técnica de edición sobre los datos.

- Estimar el desempeño esperado para el conjunto de test.

1.3. Descripción del Espacio de Características

Las observaciones en la base de datos cuentan con 54 características muy diversas. Como se detallará, algunas de ellas relevan aspectos geométricos del terreno y se miden en valores continuos. Otras toman valores discretos para representar el nivel de sombra en determinado horario. Por último se cuenta con 4 características binarias que indican el área en la Fig.1 y un conjunto de 40 características que describen el tipo de suelo.

A continuación se presenta un detalle del espacio de características disponibles:
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 (valor de 0 a 255): Índice de sombra a las 9am, en el solsticio de verano.
Hillshade_Noon (valor de 0 to 255): Índice de sombra al mediodía, en el solsticio de verano.
Hillshade_3pm (valor de 0 to 255): 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 (4 valores binarios, uno por área): Nombre del área en la que está situado.
Soil_Type: Tipo de suelo (hay 40 disponibles, una característica con valor binario por tipo de suelo).
Cover_Type: Tipo de bosque, que es la clase a estimar. Hay 7 posibles tipos.

2. Clasificación con Conjunto de Entrenamiento

2.1. Desempeños

Tomando la base de datos original se propone hacer un primer sondeo del problema probando distintos tipos de clasificadores y analizando los resultados obtenidos.

En primer lugar se observa que una buena medida del desempeño de los clasificadores será el *Accuracy* (Porcentaje de muestras bien clasificadas), debido a que el conjunto de entrenamiento está bien balanceado, es decir que hay una cantidad similar de muestras por clase, como se muestra en la Fig.2. De no estarlo, algunos clasificadores podrían clasificar con mayor peso a clases con mas cantidad de muestras.

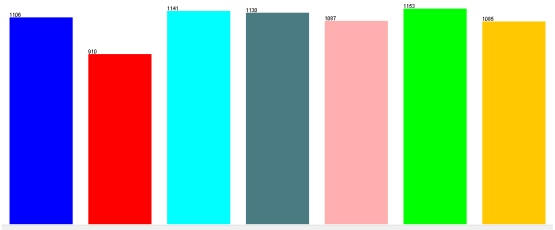


Figura 2: Cantidad de Muestras por Clase en Set Entrenamiento

Por otro lado, para corroborar la significancia estadística de los resultados (esto evita que algún valor obtenido sea azaroso o poco fiable), se entrena 5 veces a cada clasificador con validación cruzada tomando 10 particiones o folds. En este sentido, los resultados de cada validación cruzada tuvieron ciertas diferencias, y se muestra el valor medio obtenido y se detalla la desviación estándar σ en cada caso. Los parámetros de todos los clasificadores elegidos fueron optimizados mediante búsquedas exhausivas con GridSearch o CVParameterSelection y se muestran a continuación:

Naive Bayes: Valores por defecto.
SVM: Costo (C) = 20. $\gamma = 1 \times 10^{-7}$.
KNN: K=1. Se prueba 1NN.
AdaBoost: Se toman 80 iteraciones y se varía el *Weak Learner* por **RandomForest** y **J48** con valores por defecto.

Los resultados obtenidos son los siguientes:

NBayes	**SVM**	**J48**	**1NN**	**ABoost/RndForest**	**ABoost/J48**
65.96 %	69.87 %	77.26 %	78.85 %	83.87 %	85.02 %
$\sigma = 1{,}4\,\%$	$\sigma = 0{,}1\,\%$	$\sigma = 0{,}3\,\%$	$\sigma = 0{,}2\,\%$	$\sigma = 0{,}2\,\%$	$\sigma = 0{,}1\,\%$

Cuadro 1: Accuracy con Cross-Validation (10 Folds) sobre Set de Entrenamiento Original

Como puede verse, los últimos dos clasificadores cumplen con el primer ob-

jetivo planteado, donde se proponía alcanzar al menos un 82 % de desempeño sobre el conjunto de entrenamiento original.

2.2. AdaBoost

En este apartado se explican aspectos generales del algoritmo AdaBoost con el fin de fundamentar por qué ha tenido un mejor desempeño sobre los demás clasificadores y en particular el uso de Weak Learners basados en árboles de decisión.

AdaBoost es un meta-algoritmo de boosting adaptativo propuesto por Freund y Shapire en 1997 [2]. Los algoritmos de Boosting engloban una de las ideas de aprendizaje automatizado mas relevantes de los últimos 20 años. Se propone combinar la salida de múltiples clasificadores débiles (*Weak Learners*) con el fin de obtener un comité performante de clasificadores, que son entrenados secuencialmente (a diferencia de los métodos de Bagging) para obtener un clasificador de prestaciones elevadas.

El algoritmo AdaBoost en particular, funciona a grandes rasgos de la siguiente manera, como muestra la Fig.3:

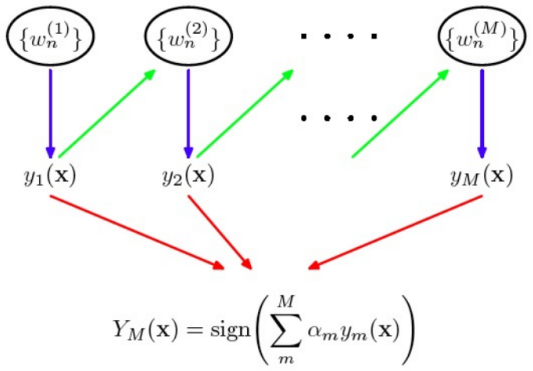


Figura 3: Ilustración Esquemática del Funcionamiento de AdaBoost

- Cada Weak Learner $y_m(x)$ se entrena usando una forma ponderada de las muestras de entrenamiento (flechas azules). Los pesos asignados $w_n^{(m)}$ dependen de los desempeños previamente obtenidos al clasificar dichas muestras con $y_{m-1}(x)$ (flechas verdes).
  El primer clasificador $y_1(x)$ se entrena con pesos uniformes.
- A las muestras mal clasificadas por un Weak Learner se le asigna un peso mayor a la hora de entrenar el clasificador del paso siguiente. Los pesos se dejan fijos si las muestras fueron bien clasificadas.
- Una vez entrenados todos los clasificadores de la cascada, sus predicciones se combinan mediante la mayoria ponderada generando el clasificador final $Y_M(x)$ (flechas rojas).

Se dice que AdaBoost es adaptativo en el sentido de que los Weak Learners subsiguientes en la cascada se ajustan de acuerdo a las muestras mal clasificadas en los pasos previos.

En este sentido, por ejemplo, es esperable que al usar AdaBoost con J48 el desempeño sea tan superior al de usar J48 sin boosting, como se ve en el Cuadro 1.

En cada iteración el algoritmo minimiza una función de error exponencial definida como:

$$E = \sum_{n=1}^{N} e^{-t_n f_m(x_n)}$$

Con $f_m(x) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(x)$ y $t_n \in \{-1, 1\}$ son los targets del conjunto de entrenamiento. $\alpha_l = Ln\{\frac{1-\epsilon_l}{\epsilon_l}\}$ donde ϵ_l representa una medida de la tasa de eror de cada Weak Learner para el conjunto de datos. Los α_l dan mayor peso a clasificadores mas precisos. De esta forma el error presenta un decaimiento rápido a lo largo de las iteraciones. Puede verse que el desempeño crece arbitrariamente con la cantidad de Weak Learners siempre que tengan un desempeño individual mayor al mínimo de los desempeños obtenidos previamente, aunque lo hace de forma logarítmica, por lo que eventualmente deja de ser provechoso por costo computacional.

Como ventajas se puede mencionar que es un algoritmo relativamente rápido, facil de programar, no requiere ajustes de parámetros previos (solo el número de iteraciones y elección de Weak Learner), es flexible dado que se puede usar con diversos tipos de clasificadores.

Una de las principales ventajas de trabajar con AdaBoost para este problema, en contraste con los otros clasificadores, es su versatilidad para trabajar con distintos tipos de características (continuas, discretas, binarias). Como contrapartida AdaBoost resulta sensible al ruido (en particular ruido uniforme) y outliers presentes en los datos.

2.3. AdaBoost + Árboles de Decisión

AdaBoost con árboles de decisión como Weak Learners es ampliamente utilizado en el estado del arte, y esto se debe a que utilizar este tipo de clasificador base presenta ciertas ventajas. En particular, varias de ellas resultan útiles para este problema:

- Los árboles de decisión son no lineales. Al hacer Boosting con métodos lineales no se obtienen buenos resultados.
- No se precisa hacer una optimización previa de parámetros (como en SVM o k-NN por ejemplo). Como los datos son ponderados con pesos en cada iteración, implicaría un costo computacional elevado optimizar estos parámetros en cada paso.

- Los árboles de decisión son relativamente rápidos de entrenar, y esto es útil dado que se construirán cientos o miles de árboles.

.

2.4. AdaBoost + RandomForest

RandomForest es un algoritmo de Bagging. Consiste en una combinación de árboles de decisión tal que cada uno de estos clasificadores se contruye a partir de un vector de muestras tomadas al azar y con la misma distribución para todos los árboles.

El error de un bosque formado por árboles de decisión clasificadores, depende del desempeño de cada árbol individualmente así como también de la correlación entre ellos. De esta forma el algoritmo reduce la tendencia a hacer overfitting de los árboles de decisión y la sensibilidad al ruido.

En la literatura se muestra como al entrenar haciendo Boosting con AdaBoost, seleccionando RandomForest como Weak Learner, se obtienen mejoras con respecto a la tendencia de overfitting y sensibilidad al ruido propias de AdaBoost.

De lo anterior se deduce que AdaBoost parece ser un clasificador apropiado para entrenar con nuestro conjunto de entrenamiento original, debido a que el espacio de características es muy diverso en cuanto a la naturaleza de las características y las distintas métricas utilizadas.

Además, Caruana et al. [1] muestran un relevamiento empírico de un conjunto de clasificadores muy variado y reporta el mejor desempeño para Boosted Decision Trees, lo que parece respaldar nuestros resultados. En consecuencia, decidimos utilizar AdaBoost+árboles de decisión.

3. Edición de Datos

3.1. Detección y Eliminación de Outliers

Es de interés visualizar en el conjunto de entrenamiento la existencia de ciertos patrones con valores atípicos dentro del conjunto de datos, comunmente denominados *outliers*. Estos patrones pueden existir debido a la existencia de ruido o error al tomar las medidas.

Se utilizó en Weka la detección de Outliers y se analizan los datos en Matlab, detecatando 87 outliers. Ninguno de ellos pertenecen a las características binarias, lo cual es razonable pues toman solo dos valores y sería extraño encontrar allí valores atípicos.

En la Fig. 4 se grafican las características que presentan valores atípicos, algunos de ellos corresponden a valores extremos en alguna de las características y otros por ser atípicos al analizar globalmente el vector de características nominales.

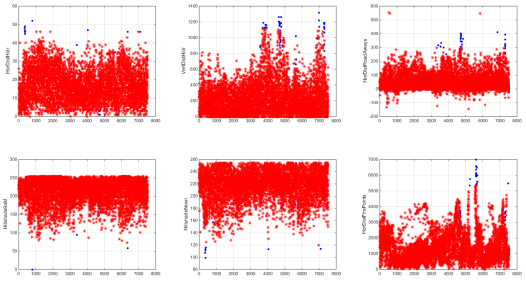


Figura 4: 87 Outliers detectados en Set de Entrenamiento Original

Al correr los algoritmos AdaBoost con 80 iteraciones y como Weak Learners RandomForest y J48 se obtienen los siguientes desempeños de clasificación para CV (10 Folds), con significancia estadística como en la sección anterior:

	ABoost/RndForest	**ABoost/J48**
Original	83.87 %	85.02 %
Sin Outliers	83.76 % $\sigma = 0{,}4$ %	85.27 % $\sigma = 0{,}7$ %

Se observa que para AdaBoost + Random Forest en general empeora el desempeño. Esto se debe a que RandomForest es poco sensible a los outliers, y al eliminarlos se pierde información que aprovechaba el clasificador.

En el caso de AdaBoost + J48 se incrementa marginalmente el desempeño, debido a que el clasificador era sensible a dichos outliers y resulta beneficioso eliminarlos del conjunto de datos; sin embargo la varianza de estos resultados es muy grande como para asegurar una mejora y decidimos no utilizar esta modificación para los resultados finales.

4. Extracción de Características

Existen diversas técnicas para modificar el espacio de características con el fin de mejorar la separabilidad de clases y favorecer las condiciones de un problema dado y así obtener un mejor desempeño de clasificación. Cumplir con dicho objetivo depende de la naturaleza del espacio de características así como del clasificador elegido. A continuación se exploran algunas de estas técnicas y su impacto en el entrenamiento.

Cabe mencionar que debido a la gran cantidad de patrones disponibles en el set de entrenamiento comparado a la cantidad de características, el problema está bien condicionado y no parecería un aspecto primordial, a priori, reducir la dimensionalidad para obtener mejoras. Esto se hace necesario cuando tenemos pocos patrones en relación al número de características. En nuestro caso se cumple con un cierto margen el criterio de buen condicionamiento del problema:

$$\frac{n}{d} = \frac{\text{cantidad de patrones por clase}}{\text{cantidad de atributos}} = \frac{7650/7}{54} \approx 20{,}28 > 10$$

En este sentido se trabajará sobre extracción de nuevas características mas relevantes que las originales, y no se utilizarán técnicas de selección de características que implicarían borrar algunos atributos y quedarse solo con los mas relevantes para reducir la dimensionalidad de los patrones.

4.1. Modificación de Atributos Wilderness_Area

Se pretende explorar el impacto de transformar las 4 características binarias a una única característica nominal que toma valores 1,2,3 y 4 indicando el Wilderness_Area.

A priori, esto podría ayudar al clasificador al mejorar la interpretación sobre los datos al existir una única característica, aunque sabemos de antemano que los árboles de decisión (nuestros Weak Learners) aprovechan los atributos binarios. Por otro lado se disminuiría la dimensionalidad y esto podría generar alguna mejora marginal.

El impacto obtenido en el desempeño de los clasificadores elegidos es el siguiente, entrenando con CV (10 Folds) y luego de 5 tiradas:

	ABoost/RndForest	**ABoost/J48**
Original	83.87 %	85.02 %
Wild_Area Nominal	83.58 % $\sigma = 0{,}3$ %	84.90 % $\sigma = 0{,}3$ %

Se concluye que esta modificación sobre el espacio de características es contraproducente para el desempeño, con respecto a clasificar con la base de datos original. Se corrobora que los árboles de decisión aprovechan mejor el hecho de clasificar con varias características binarias en vez de una única nominal.

4.2. Modificación de Atributos Soil_Type

4.2.1. Soil-Types de Binario a Nominal

A pesar de la conclusión anterior, en la que se ve que los árboles clasifican mejor atributos binarios que nominales, se prueba el impacto de convertir las 40 características correspondientes al tipo de suelo, a una única característica nominal.

Los resultados obtenidos son los esperados. Nuevamente con CV (10 Folds), se obtiene un empeoramiento en el desempeño de clasificación, aún más notorio que en el caso anterior:

	ABoost/RndForest	**ABoost/J48**
Original	83.87 %	85.02 %
Soil_Type Nominal	83.11 % $\sigma = 0{,}3$ %	84.66 % $\sigma = 0{,}2$ %

4.2.2. Merge de Soil-Types Similares

Debido a la existencia de distintos tipos de suelos que presentan características muy similares (algunos de ellos son prácticamente idénticos), se prueba la idea de combinar ciertos Soil_Types haciendo un simple OR entre dichas columnas en la base de datos. Fusionamos características de familias iguales pero que difieren en ser "very stony" o solo "stony", por ejemplo.

Esto permite eliminar redundancia en los datos y evitar clasificar con Soil_Types irrelevantes, a la vez que se disminuye la dimensionalidad de los patrones. Se modifico la base haciendo Merge con los siguientes Soil_Types:

- {2,4,6}
- {22,23,24,25,26,27,28,31,33,38}
- {10,11}
- {19,20,21}
- {29,30}
- {35,37}
- {39,40}

A su vez se eliminan las características Soil_Type 15, 7 y 8, dado que no existe ningun patrón en todo el conjunto de entrenamiento que tenga dicho tipo de suelo.

Los resultados obtenidos, nuevamente con CV (10 Folds), se muestran en la siguiente tabla. Se obtiene un empeoramiento en el desempeño de clasificación en ambos casos:

	ABoost/RndForest	**ABoost/J48**
Original	83.87 %	85.02 %
Merge de Soil_Type	82.88 % $\sigma = 0{,}2$ %	84.67 % $\sigma = 0{,}3$ %

Probablemento este resultado se debe a que eliminamos información que no era tan redundante como parecía a simple vista, y AdaBoost utilizaba estos datos para disminuír el error de clasificación.

4.2.3. Soil_Types Desarmado en Nuevo Espacio de Atributos

Ahora tenemos presente que para los clasificadores elegidos no es una buena opción modificar las características binarias y transformarlas en nominales, así como hacer merge de características para disminuír cierta redundancia presente en los datos.

Sin embargo, está a la vista que el formato de las características de SoilTypes carece de interpretación para cualquier clasificador, dado que engloba una serie de observaciones muy diversas sobre los terrenos, y datos sobre la familia del tipo de bosque, siendo estos datos independientes y sin ninguna estructura aparente a lo largo de los 40 Soil_Types presentes.

Por ejemplo Soil_Type 1 indica: Cathedral family + Rock outcrop complex + extremely stony.
Soil_Type 2 por otro lado indica: Vanet + Ratake families complex + very stony.

Proponemos desarmar este conjunto de 40 características con el fin de conformar un nuevo conjunto con mayor poder de interpretación sobre los datos, con el fin de explorar la repercución que tiene en los clasificadores.

No se elimina ninguna información. Cada propiedad que aparece en un Soil_Type determinado se transforma en una nueva característica, y se establece un valor discreto en el rango 1, 2 y 3 de acuerdo a si está propiedad está catalogada como *extremely* (3), *very* (2), o si no se indica nada (1).

Como resultado de esta transformación se obtienen 23 características a partir de las 40 originales, y los desempeños obtenidos con CV (10 Folds) para varias tiradas se muestran en la siguiente tabla:

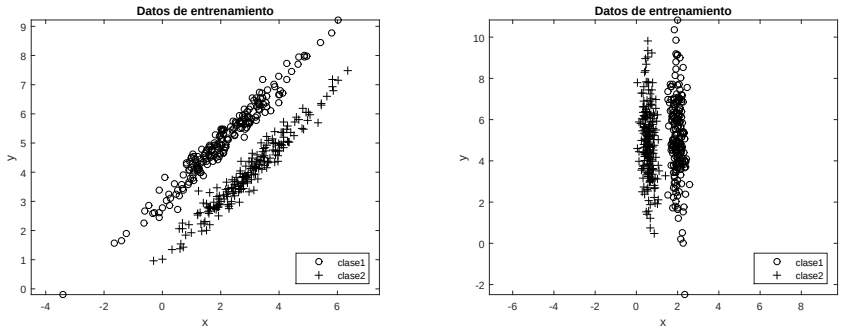
	ABoost/RndForest	**ABoost/J48**
Original	83.87 %	85.02 %
Soil_Types Desarmados	83.23 % $\sigma = 0{,}3$ %	85.09 % $\sigma = 0{,}3$ %

Se observa una mejora marginal solo en caso de utilizar como Weak Learner el algoritmo J48, pero la varianza del resultado tampoco nos permite validarlo como para todos los casos.

4.3. PCA sobre Características Correlacionadas

Los árboles de decisión funcionan mejor con regiones de decisión rectangulares, pues en un nodo ponen umbrales para una característica sola. Por ello,

exploramos el efecto de aplicar PCA y LDA, que además de reducir dimensionalidad, logran hacer una transformación que favorece regiones de decisión rectangulares. El ejemplo más claro es el siguiente, donde PCA halla la dirección de máxima varianza y, con ello, vuelve la frontera de decisión mucho más sencilla de generar con un árbol.



(a) Datos originales (b) Datos después PCA

Figura 5: Resultado de aplicar PCA

Además, observamos que existen características que deberían estar correlacionadas. En este caso, puede utillizarse PCA para sintetizar características nuevas no correlacionadas y eliminar redundancia.

En primer lugar, las características Elevation y Vertical Distance to Hydrology están obviamente correlacionadas porque miden distancias verticales.

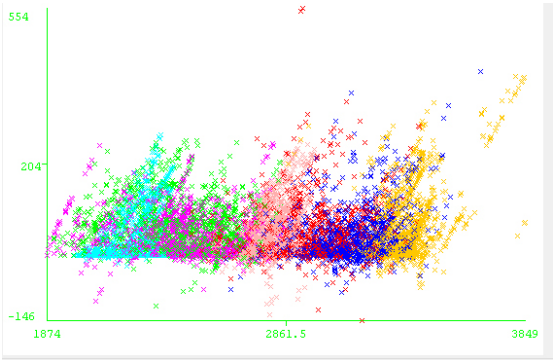


Figura 6: Correlación entre Elevation y Vertical Distance to Hydrology

Por otro lado, esperamos que las características Slope, Aspect y las tres Hillshades estén correlacionadas. Esto se debe a que, dada una orientación del suelo en el espacio (Slope y Aspect) y dada la iluminación a cierta hora del día (Hillshade9am, por ejemplo), la iluminación a cualquier otra hora del día queda determinada por la trayectoria del sol. Esto varía según la época del año, pero aún así es esperable ver una cierta correlación.

Con esto en mente, proponemos las siguientes extracciones de características y sus resultados:

- PCA a Slope, Aspect y Hillshades. Las demás quedan iguales: 85.46 %. $\sigma = 0{,}5\,\%$
- PCA a Elevation y Vertical distance to hydrology: 84.68 % $\sigma = 0{,}2\,\%$.
- LDA a Elevation y Vertical distance to hydrology. PCA a la nueva característica de LDA y a Slope, Aspect y Hillshades: 85.25 % $\sigma = 0{,}3\,\%$
- Idem anterior pero cambiando LDA por una transformación manual: 85.75 % $\sigma = 0{,}2\,\%$

En todos los casos tomamos las primeras dos componentes de PCA pues se observa en el espectro de valores propios que allí se concentra la gran mayoría de la varianza total.

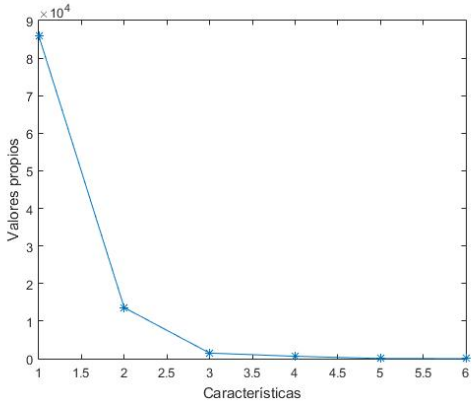


Figura 7: Espectro de valores propios. Tomamos 2 componentes.

Observamos que PCA no logra una buena transformación de las características Elevation y Vertical distance to Hydrology, a pesar de su correlación, pues el desempeño baja a 84.68 %. Por ello, probamos con LDA que tiene en cuenta la separabilidad entre clases, y se observa una pequeña mejora.

Sin embargo, al observar la transformación obtenida con LDA, no es del todo ideal. El resultado es casi una rotación de 90°, que no cambia mucho el problema. Para resovler esto, aprovechamos el hecho de que es una transformación en dos dimensiones para hacerla manualmente. La transformación implementada es una rotación de 45° y una resta de la recta $y = x$. Esta operación produce la nueva característica que se ve en la figura 8, que mejora bastante la separabilidad de las clases. Además, es la extracción que obtiene mejor desempeño.

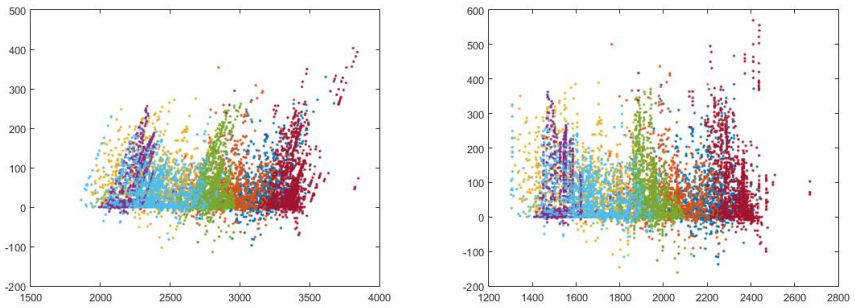


Figura 8: Resultado de aplicar rotación y traslación

5. Matriz de confusión

A continuación estudiamos la matriz de confusión reslutante al aplicar la extracción de características que obtuvo mejor desempeño con el clasificador AdaBoost+J48. Esto es útil para ver dónde están los puntos débiles del clasificador y ver cómo puede mejorarse.

```
  a   b   c   d   e   f   g   <-- classified as
828 172   1   0  25   4  76 |   a = 1
181 568  25   0  91  35  10 |   b = 2
  0  12 944  47  11 127   0 |   c = 3
  0   0  22 1096  0  12   0 |   d = 4
  7  37  11   0 1018 14   0 |   e = 5
  2  12 123  28   5 983   0 |   f = 6
 43   1   1   0   0   0 1040 |   g = 7
```

Puede verse que el error global está gobernado por la confusión entre las clases 1 y 2, así como las clases 3 y 6. Para resolver el problema, proponemos modificar el clasificador juntando las clases confusas en superclases (llamamos a 1 y 2 la clase 12 y a 3 y 6 la clase 36). Luego, las muestras que sean clasificadas como clase 12 son separadas en 1 y 2 por otro clasificador SVM y análogamente para 3 y 6.

En primer lugar, observamos que el desempeño mejora radicalmente al juntar las clases confusas, pues AdaBoost+J48 obtiene 93.9438 % en Weka y la matriz de confusión se vuelve mucho más diagonal:

```
   a    b    c    d    e   <-- classified as
1799   64    0   90   63 |   a = 12
  20 2199   58   17    0 |   b = 36
   0   42 1088    0    0 |   c = 4
  49   24    0 1013    1 |   d = 5
  30    1    0    2 1052 |   e = 7
```

Figura 9: AdaBoost+J48, CV, clases fusionadas: 1 y 2, 3 y 6.

Al entrenar el clasificador SVM con GridSearch para seleccionar C y γ, obtenemos $C = 20$ y $\gamma = 1 \times 10^{-7}$. Sin embargo, hallamos que se vuelve muy difícil mejorar el desempeño del clasificador original restringido a las dos primeras clases. Observando la matriz de confusión, el desempeño en las clases 1 y 2 con AdaBoost+J48: $acc = (828 + 568)/(828 + 568 + 172 + 181) = 79{,}81\,\%$. Tras entrenar el SVM obtenemos 79,31 %, por lo que la performance global de hecho baja. Incluso asumiendo que se pudiera obtener un desempeño del SVM por sobre el del clasificador original, debería ser significativamente mayor para que redundara en una ganancia de performance global. Esto se debe a que si los clasificadores están en cascada, sus errores se empiezan a multiplicar: basta que se equivoque uno para que el resultado sea incorrecto, pues el siguiente en la cascada no puede corregir. El SVM tiene que lidiar con un 7 % de muestras mal clasificadas (el clasificador de la primera etapa tiene un 93 % de acierto) que constituirán un error, sin importar que tan bien se entrene.

Finalmente, concluimos que las clases 1 y 2 no son separables y es muy difícil lograr un desempeño lo suficientemente mejor al que ya teníamos originalmente para que sea justificable este sistema. Además, observamos que la cascada de clasificadores no es una buena solución en general, pues propagan los errores. Es por ello, en parte, que han tenido éxito las técnicas de combinación como boosting o bagging, que combinan clasificadores horizontalmente y no secuencialmente.

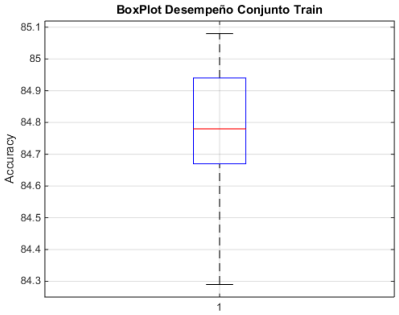
6. Clasificación con Conjunto Test

Finalmente se entrenó el clasificador con el algoritmo AdaBoost con árboles de decisión como Weak Learner implementado en Matlab, para mejorar la velocidad de entrenamiento y facilitar la extracción de características. Los árboles utilizados para AdaBoost en Matlab se toman con un mínimo de 4 instancias por hoja, la decisión de partir un nodo se toma por entropía y se poda por impurza. En el entrenamiento se toman 80 Weak Learners para AdaBoost.

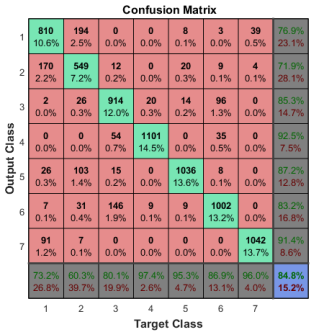
Se decidió hacer las modificaciones sobre las características que obtenían un desempeño estadísticamente significativo. Esto es:

- Transformación manual a Elevation y Vertical distance to hydrology
- PCA a la nueva característica del paso anterior, Slope, Aspect y Hillshades

Esto da un desempeño del 84.78 % $\sigma = 0{,}24\,\%$ sobre el Set Entrenamiento CV (10 Folds) en 10 tiradas.



(a) Desempeño Set Entrenamiento



(b) Matriz Confusión Set Entrenamiento

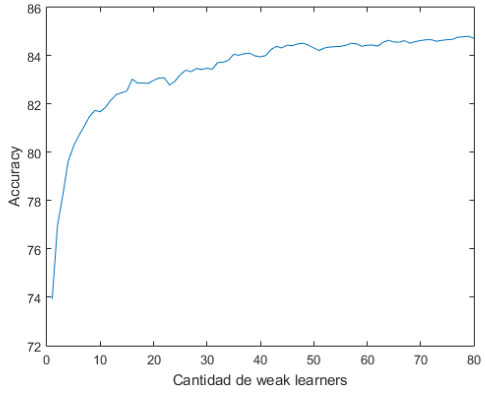


Figura 11: Accuracy AdaBoost al iterar

Se midió el desempeño obtenido por el clasificador pero esta vez sobre el conjunto de test disponible. Dado que el desempeño parece seguir creciendo, aunque

marginalmente, con la cantidad de Weak Learners, se toma el test con 150 árboles boosteados. Los resultados obtenidos y la matriz de confusión obtenida se muestran a continuación:

ABoost/J48	**Train**	**Test**
Accuracy	84.78 %	86.22 %
Std.Dev σ	0.24 %	

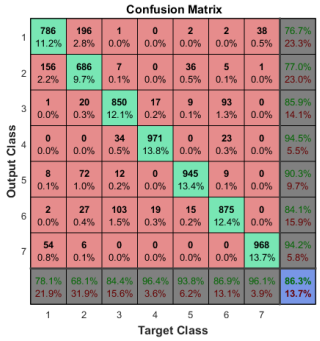


Figura 12: Matriz de Confusión Conjunto Test

7. Conclusiones

Se cumplió con el primer objetivo del proyecto de forma sencilla al utilizar el algoritmo de boosting AdaBoost con árboles de decisión como Weak Learner, que obtienen un 85.02 % de desempeño con los parámetros por defecto en Weka.

La dificultad se concentró en la extracción de nuevas características más relevantes, que ayudasen a la separabilidad de las clases. Se probaron diversas técnicas con las cuales se esperaba obtener mejoras en la performance del clasificador, pero muchas veces los resultados no fueron los esperados y se descartaron dichas modificaciones.

Las técnicas que sí dieron resultado implicaron hacer transformaciones geométricas manuales para poder discriminar las clases linealmente y hacer combinación lineal de características que presentan correlación (PCA), utilizando la información a priori del problema.

Por otro lado, se intentó resolver el problema de confusión de clases particulares implementando un dicotomizador SVM en cascada con un clasificador inicial con clases fusionadas, pero el desempeño fue claramente inferior. Se observó que la configuración en cascada de clasificadores en general no es una buena solución y la combinación horizontal como bagging o boosting es mucho mejor.

Finalmente, se observó que el desempeño en el conjunto test fue bien predicho por el cross validation, tanto por el accuracy como por las matrices de confusión, y se logró sobre un 86 % de acierto.

Referencias

[1] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[2] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.