Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tópicos de Minería de Datos

Trabajo Práctico 4: Métodos supervizados avanzados

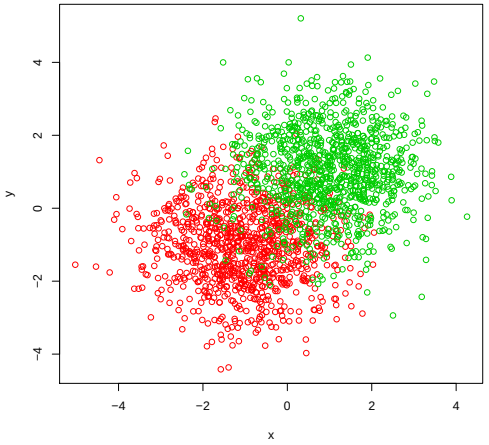
Alumno: Jeremías Rodríguez
Profesor: Pablo Granitto
22 de enero de 2018

1. Ejercicio 1: Boosting

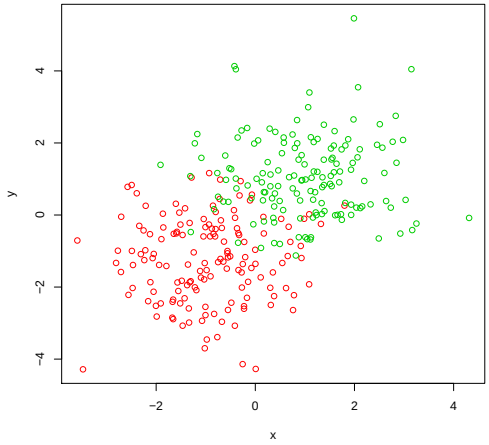
En este ejercicio analizaré distintos aspectos del método Boosting aplicado a los datasets Diagonal y Espirales Anidadas. Se utilizarán árboles de decisión como weak lerners, y se analizará cómo varía el error de acuerdo a su complejidad. Adicionalmente, analizaré brevemente cómo influye la cantidad de weak lerners utilizados.

1.1. Dataset Diagonal

Comenzaré aplicando boosting al dataset diagonal, que consiste en dos gaussianas con un cierto solapamiento:

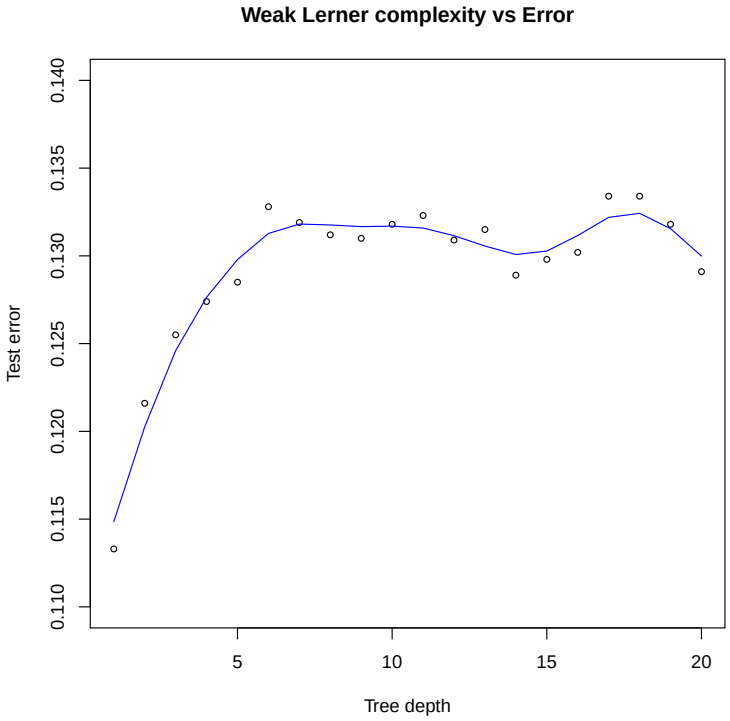


(a) Conjunto de test, dataset diagonal



(b) Conjunto de training, dataset diagonal

Para estudiar cómo funciona boosting con weak lerners de distintas complejidades, se calculó el error en test restringiendo la profundidad máxima de los árboles desde 1 hasta 20. Para cada profunidad dada, se corrió el algoritmo 10 veces y se promedio el error en test, generando la siguiente gráfica.

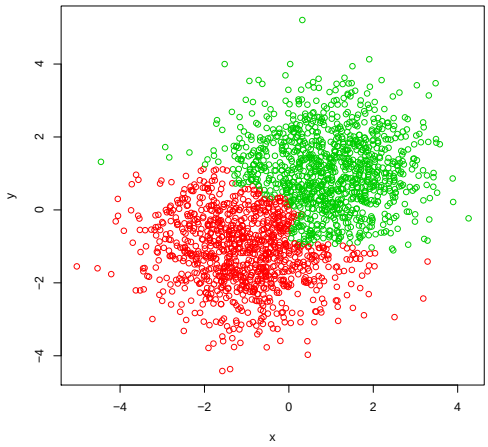


Se puede ver que, al aumentar la complejidad del weak lerner, el error aumenta ligeramente. Por lo tanto, el problema diagonal se beneficia más de boosting utilizando clasificadores simples. ¿Por qué?

Cuando usamos un weak learner con profundidad=1 (stumps), hacemos cortes sencillos. Por ejemplo, el primer corte de una cierta corrida es:

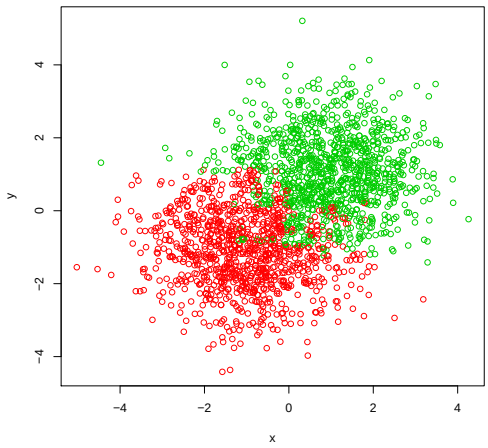
```
1) root 300 142 1 (0.4733333 0.5266667)
  2) y < -0.09284498 138  18 0 (0.8695652 0.1304348) *
  3) y >=-0.09284498 162  22 1 (0.1358025 0.8641975) *
```

Luego de realizar 200 de estos cortes simples, la superficie de decisión final tiene más o menos la siguiente forma:



Como vemos, se divide claramente a las dos gaussianas con una frontera escalonada y sencilla. Varios puntos quedan mal clasificados si comparamos con la clasificación real, pero la división tiene mucho sentido pues más o menos asigna los puntos a la gaussiana mas cercana.

En cambio, si usamos un weak lerner muy complejo, con profunidad=20. La superficie de decisión final tiene la suiente forma:

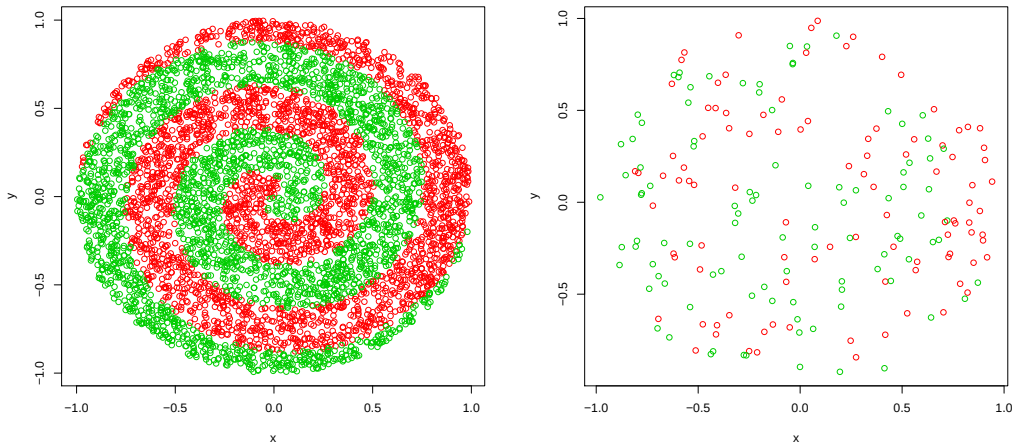


La clasificación obtenida es algo más compleja, y ya no hay una frontera sencilla de división. Por ejemplo, ciertos sectores que están más proximos a la gaussiana inferior son asignados a la gaussiana superior. Esto podría suceder porque al usar clasificadores mas complejos, boosting permitiría representar fronteras mucho mas complejas que tal vez intentan acomodar la clasificación al conjunto de training y no generalizan tan bien (overfitting).

Por lo tanto, este problema se ve (levemente) beneficiado por clasificadores sencillos que permitan separar claramente las dos gaussianas sin crear fronteras muy elaboradas para ajustar puntos solapados de training.

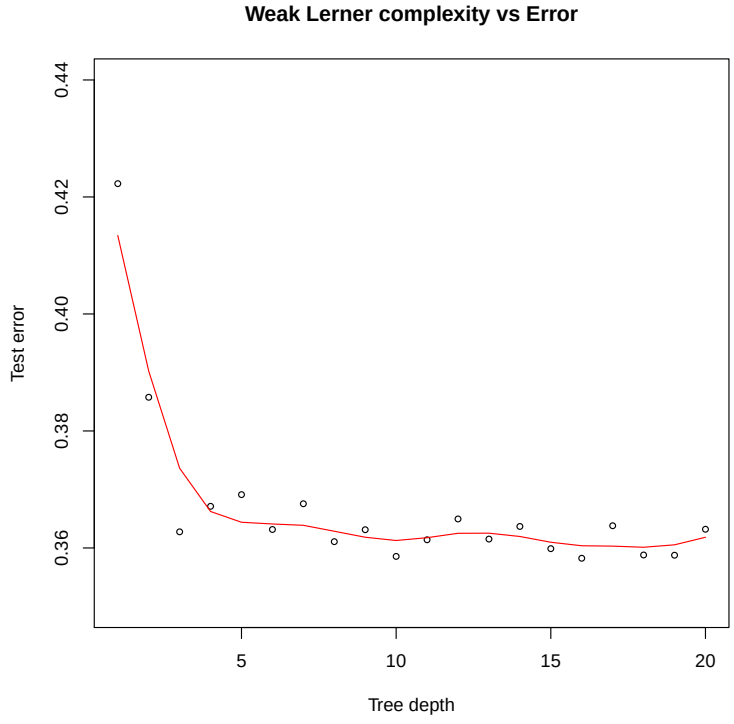
1.2. Dataset Espirales Anidadas

A continuación se muestran los datasets de test y training utilizados. Notar que el dataset de training tiene ruido y es bastante difícil distinguir que está representando dos espirales.



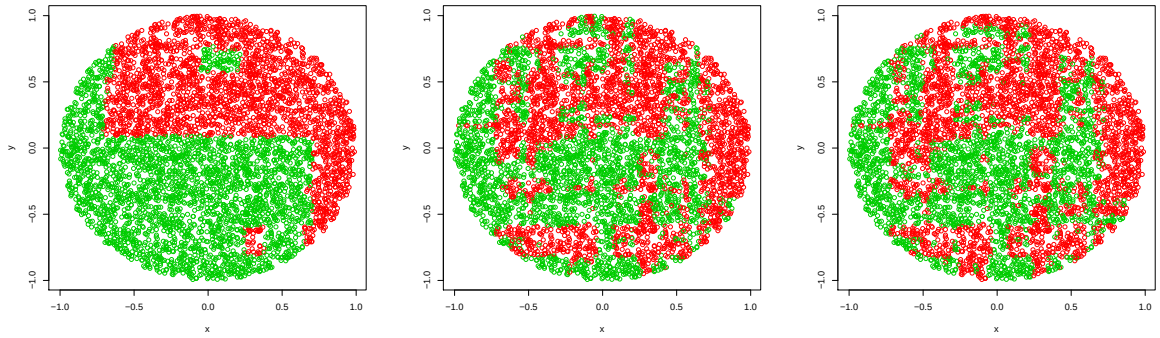
(a) Conjunto de test, espirales anidadas (b) Conjunto de training, espirales anidadas

Al igual que para el dataset diagonal, se calculo el promedio del error utilizando árboles con profundidades desde 1 hasta 20, obteniendo la siguiente gráfica:



En este caso, se ve que aumentar la complejidad del weak lerner mejora el error, aunque nuevamente la diferencia de utilizar clasificadores más complejos es ligera. Se ve que el error es mucho mayor al obtenido en el dataset diagonal, esto podría deberse a que es más complejo aprender a clasificar bien las espirales entrenando con un dataset con pocos puntos y con ruido.

Para intentar explicar porqué en este caso boosting funciona mejor con weak lerners más complejos,veamos las siguientes predicciones:



(a) Predicción test. Depth = 1 (b) Predicción test. Depth = 10 (c) Predicción test. Depth = 20

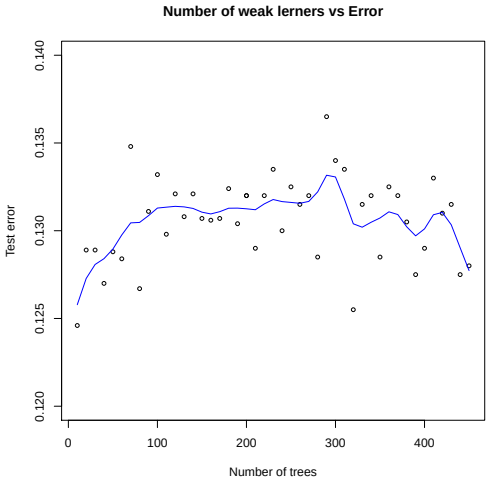
Al tener que aprender fronteras muy complejas, como las espirales, aparentemente boosting se ve un poco más beneficiado de usar weak lerners más complejos, como el siguiente:

```
1) root 200 96 1 (0.48000000 0.52000000)
  2) x>=-0.6323627 171 78 0 (0.54385965 0.45614035)
    4) y>=0.09143471 81 25 0 (0.69135802 0.30864198)
      8) y< 0.4183791 24  2 0 (0.91666667 0.08333333) *
      9) y>=0.4183791 57 23 0 (0.59649123 0.40350877)
       18) y>=0.6929055 21  2 0 (0.90476190 0.09523810) *
       19) y< 0.6929055 36 15 1 (0.41666667 0.58333333)
         38) y< 0.5926814 23  9 0 (0.60869565 0.39130435)
           76) y>=0.5038303 12  2 0 (0.83333333 0.16666667) *
           77) y< 0.5038303 11  4 1 (0.36363636 0.63636364) *
         39) y>=0.5926814 13  1 1 (0.07692308 0.92307692) *
    5) y< 0.09143471 90 37 1 (0.41111111 0.58888889)
     10) x< -0.464413 10  2 0 (0.80000000 0.20000000) *
     11) x>=-0.464413 80 29 1 (0.36250000 0.63750000)
       22) x>=-0.1958606 58 27 1 (0.46551724 0.53448276)
         44) x< -0.066903 12  2 0 (0.83333333 0.16666667) *
         45) x>=-0.066903 46 17 1 (0.36956522 0.63043478)
           90) x>=0.724386 12  3 0 (0.75000000 0.25000000) *
           91) x< 0.724386 34  8 1 (0.23529412 0.76470588) *
       23) x< -0.1958606 22  2 1 (0.09090909 0.90909091) *
  3) x< -0.6323627 29  3 1 (0.10344828 0.89655172) *
```

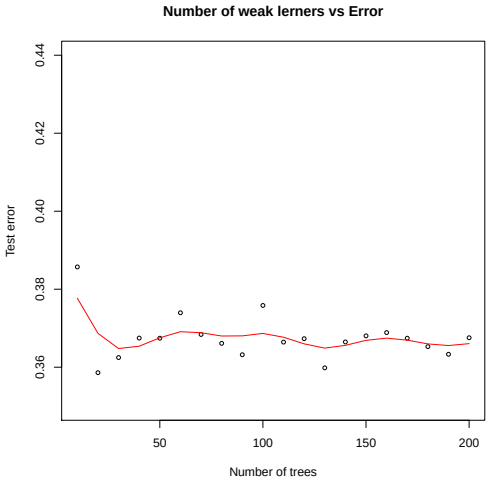
De este modo, las fronteras adoptan formas más sofisticadas y el error mejora un poco.

1.3. Cantidad de weak lerners

A modo de curiosidad, también probé qué sucede si se varía la cantidad de weak lerners utilizados en el ensemble. En un principio pensé que, si la cantidad de weak lerners es muy grande, debería producirse overfitting. Sin embargo, fijando depth=5 para ambos datasets, obtuve el siguiente resultado:



(a) Dataset diagonal



(b) Dataset espirales anidadas

Las gráficas parecen indicar que el error varía poco aunque se aumente mucho la cantidad de weak lerners. Es decir, el error parece estabilizarse.

2. Ejercicio 2

En este ejercicio se aplicaron los métodos random forest, boosting y svm al dataset Lampone. Este dataset consta de 49 filas y 144 features referentes a arándanos, en el cuál la variable target es la Especie de Arándano. Antes de poder aplicar los métodos, se descartaron columnas no numéricas y no válidas.

Luego de preprocesar el dataset, se realizó una estimación del error en 5-folds. Cabe resaltar que, como el dataset tiene tan pocas muestras, el error varía mucho según la elección aleatoria del fold.

La siguiente tabla resume el mejor resultado de cada método:

Método	Error
Random Forest	0.1555556
Boosting	0.11111
SVM Polinomial	0.08888889
SVM RBF	0.1777778

2.1. Random Forest

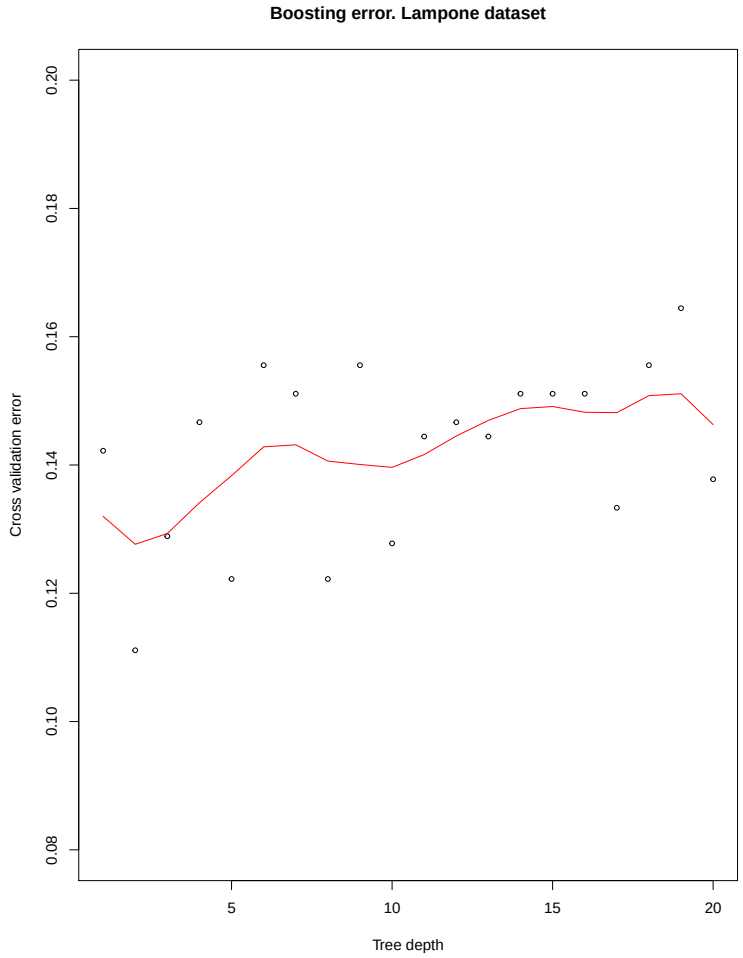
Este método fue el más sencillo de aplicar. No hubo que ajustar parámetros, y el resultado se generó muy rápido. El error obtenido fue de 0.1555556.

2.2. Boosting

Por otro lado, boosting fue el método más complejo de utilizar. Fijé una cantidad de árboles n=100, y luego estimé el mejor parámetro depth usando grid search con depth entre 1 y 20.

Dada la aleatoriedad del método, para cada depth fija realicé 5 corridas de boosting y promedié el resultado. El tiempo total de ejecución fue mayor a 10 horas, lo cual evidencia una desventaja del método.

La siguiente gráfica muestra el error en función de la profundidad:



Como podemos ver, el valor óptimo es 0.11111 para depth=2 y la tendencia parece ser creciente. Es decir, lampone se beneficia más de usar árboles sencillos. Sin embargo los resultados son muy dispares, y el resultado podría ser simple casualidad por la elección inicial de los folds (el dataset tiene muy pocos puntos).

2.3. SVM

Finalmente, apliqué el método SVM. Aunque no fue tan complicado como boosting, este método también tiene la desventaja (en comparación con random forest) de que hay parámetros que optimizar: el valor del costo C, la elección del kernel y los parámetros de este.

Se realizó la optimización para los dos kernels más populares: polinomial y gaussiano.

2.3.1. Kernel polinomial

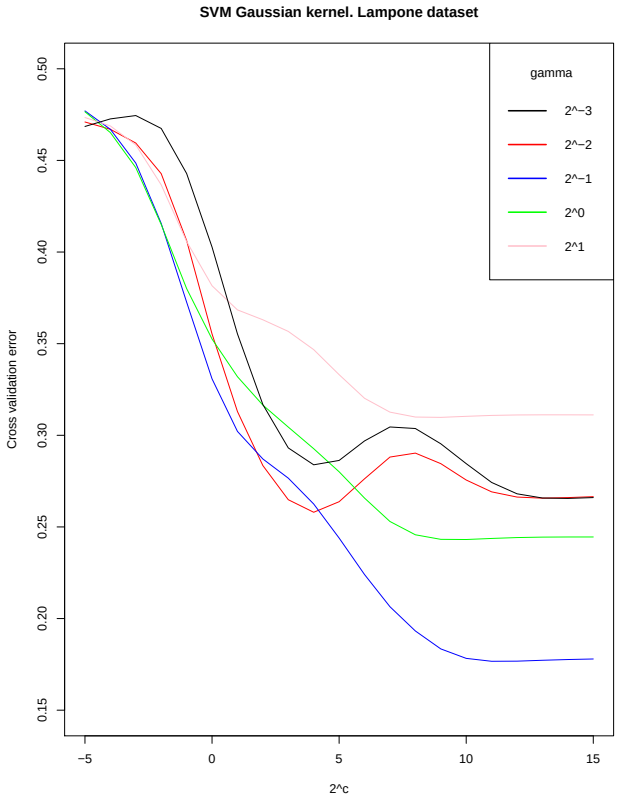
Optimicé los parámetros C (en un rango de $2^{-5} - 2^{15}$) y grado del polinomio n (en 1:5). La gráfica en la siguiente hoja muestra el error obtenido en función de C, para cada grado.

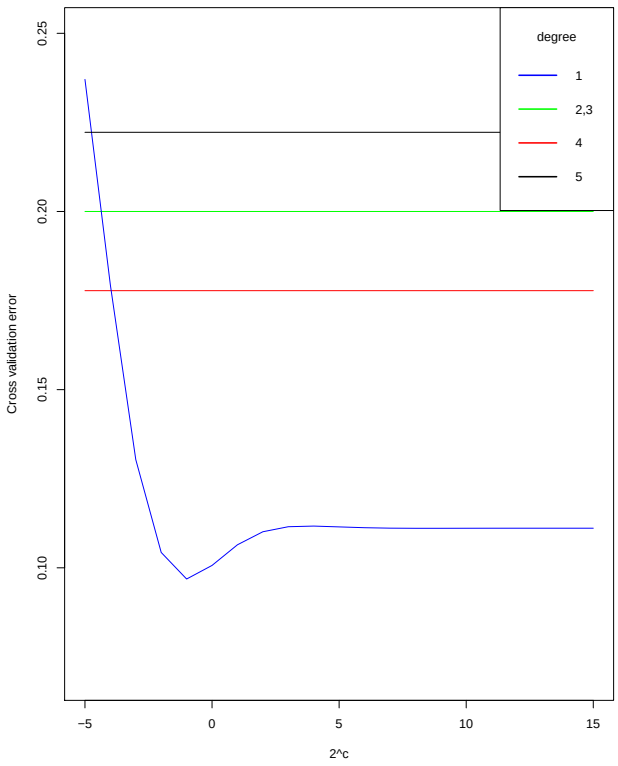
El óptimo se encontró usando un kernel lineal, con $c = 2^{-3}$ y un valor de 0.08888889

2.3.2. Kernel gaussiano

Para este kernel, optimicé los parámetros C (en un rango de $2^{-5} - 2^{15}$) y γ (en un rango de $2^{-15} - 2^{3}$)

El óptimo se encontró para $\gamma = 2^{-1}$ y cualquiera de los valores más grandes de c, siendo el error de 0.1777778.


SVM Gaussian kernel. Lampone dataset
gamma
2^−3
2^−2
2^−1
2^0
2^1
Cross validation error
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
−5
0
5
10
15
2^c


SVM polinomial kernel. Lampone dataset
degree
1
2,3
4
5
Cross validation error
0.10
0.15
0.20
0.25
−5
0
5
10
15
2^c

2.4. Conclusiones y análisis

Los tres métodos arrojaron resultados muy buenos, considerando que el dataset tiene muy pocas muestras y muchas dimensiones. Como la dimensionalidad es tan alta, necesitaríamos tener muchos ejemplos de training para clasificar con mayor precisión. A continuación analizaré brevemente el resultado de cada método:

- Random Forest: Como hemos estudiado en machine learning, los árboles de decisión son proclives a realizar overfitting en datasets de alta dimensionalidad. Por ejemplo, un sólo arbol arroja el siguiente error: 0.2666667. La combinación de varios de esos árboles en el algoritmo de random forest mejora este número, generando un clasificador razonablemente bueno en muy poco tiempo. Sin embargo, el resultado fue el peor de todos.
- SVM: Los mejores resultados se consiguieron rápidamente con SVM lineal. Aparentemente, las SVM funcionan bien ante datasets de muchas dimensiones. Pienso que SVM lineal funciona mejor porque, al haber tantas features y pocos puntos, claramente estos deben ser linealmente separables por un hiperplano y no hay necesidad de utilizar un kernel.
- Boosting: El resultado de boosting es bueno, a pesar del conjunto de entrenamiento tan pequeño. El tiempo de entrenamiento para hallar los parámetros óptimos fue aproximadamente 10 horas, sin embargo el algoritmo es muy lento para datasets angostos también.