**Aprendizaje automático 2019**

# Clasificación de facies geológicas utilizando aprendizaje automático supervisado

Alumno: Gabriel Ricardo Gelpi
Docente: Franco Ronchetti

## Objetivos e introducción

El objetivo de este trabajo es mostrar el alcance y utilidad de las técnicas de clasificación estudiadas en el curso de Aprendizaje automático en el contexto de las geociencias. Para esto se parte de un tutorial (1) donde se realiza la clasificación de facies sedimentarias a partir de datos de pozos exploratorios mediante Support Vector Machine, y se compara los resultados obtenidos con esta técnica con los resultados de los algoritmos abordados durante el curso. La determinación de las facies sedimentarias extraídas de pozos exploratorios es de gran utilidad para poder construir la historia geológica de una cuenca sedimentaria, y así por ejemplo, poder definir el sistema petrolero para la exploración y producción de hidrocarburos (petróleo y gas). El set de datos que se utiliza es provisto por la Universidad de Kansas (USA). Como librería de herramientas de aprendizaje automático se utiliza *Scikit-learn* (https://scikit-learn.org/stable/).

## Datos: Origen, exploración y acondicionamiento de los datos

Los datos pertenecen al campo de gas Hugoton ubicado en Kansa, Estados Unidos. El data set consiste de 5 perfiles de distintas propiedades físicas medidos en cada pozo y dos indicadores de facies (7 features). Dado que esta es una región ya estudiada se cuenta con un "label" o etiqueta con las facies determinadas a partir del estudio de las coronas del pozo. Hay 9 tipo de facies (numeradas de 1-9) identificadas. En este contexto, las técnicas de aprendizaje automático a utilizar son las conocidas como "SUPERVISADO".

Se separa uno de los pozos para evaluar/testear al final (Shankle well) los modelos entrenados con los distintos algoritmo.

Como es sabido, varios algoritmos de machine learning requieren que los datos se encuentren estandarizados. Para esto se utiliza el paquete *StandardScalar* de Scikit-learn. Se utilizó *train_test_split* para separar los datos en un grupo de entrenamiento y otro de testeo del algoritmo de clasificación con los parámetros seleccionados. En este caso se eligió un 80% para entrenar y 20% para testear los clasificadores.

## Entrenando los clasificadores

Para la clasificación de las facies se utilizan 4 algortimos: K-nearest neighbor (KNN), Redes neuronales (RN), Logistic Regresión (LR) y Support Vector Machine (SVM). Los primero 3 fueron vistos en clase. La idea de SVM es separar los datos mediantes rectas o planos. Esto no es tan trivial de hacer en la práctica, no siempre se puede separar los datos por planos perfectos, por lo que SVM se apoya en el uso de funciones (kernels) que le permiten proyectar los datos en

espacios de altas dimensiones donde sí pueden ser separados. Al igual que los otros tres algoritmos, *Scikit-learn* tiene implementado Support Vector Machine.

Es importante considerar que estos algoritmos tienen un número importante de parámetros que deben de ser elegidos. Estos pueden afectar la performance del clasificador y por lo tanto la decisión a tomar respecto del problema que se quiere resolver.

En el código adjunto se puede ver como se realiza el entrenamiento de los clasificadores utilizando *Scikit-learn*. Luego se analiza con los datos de testeo la calidad de las clasificaciones.

Lo primero fue utilizar todos estos algoritmos con sus parámetros por default. En la Figura 1 puede observarse las distintas métricas para cada facie. Las métricas son Precision, Recall y F1 score. La precisión nos dá la probabilidad de que una facie clasificada realmente pertenezca a esa clase. El Recall es la probabilidad de que una muestra sea bien clasificada para una clase dada. F1 es una combinación de ambas métricas que busca un balance entre ambas medidas. Se busca los parámetros que lleven, en lo posible, a valores altos principalmente del Recall y F1. También en las Figuras 2, 3, 4 y 5 pueden verse las matrices de confusión de cada técnica. Estas matrices nos permiten ver de manera rápida como está funcionando el modelo. Lo principal, para un primer análisis, es la diagonal principal.

La siguiente tabla muestra estas métricas promedio para cada método.

| | Knn | Lr | Red | SVM |
|---|---|---|---|---|
| Precision | 0.7 | 0.55 | 0.69 | 0.64 |
| Recall | 0.69 | 0.52 | 0.68 | 0.55 |
| F1 | 0.69 | 0.52 | 0.68 | 0.56 |

Puede verse que Knn y las Redes tienen la mejor performance. Sin embargo, al mirar estas métricas para cada facie se observa que no es tan claro cual algoritmo es el óptimo para realizar la clasificación de facies. Mirando las matrices de confusión también puede observarse esto prestando atención a los valores de la matriz diagonal (los aciertos) y lo que pasa fuera de la misma. Los cuales deberían de ser valores nulos o lo mas bajos posibles.

El siguiente paso para mejorar estos resultados fue seleccionar los parámetros para los distintos algoritmos. En el tutorial puede verse el análisis para la selección de los parámetros propuestos para el caso de SVM. Si bien *Scikit-Learn* tiene varias técnicas para la selección (Tuning) de los parámetros, en este trabajo el análisis fue a prueba y error (por lo que queda margen para mejorar*). En el caso de KNN, solo se muestran los ejemplos de k igual a 3 y 8. Para las redes neuronales se utilizaron distintos números de capas con distinto número de neuronas. Se muestran los casos de 100, 50 y el otro de 100, 50 y 10. En la Figura 6, 7 y 8 puede verse la performance de cada algoritmo con los distintos parámetros de cada algoritmo. Una mejora en los valores de las métricas individuales y de forma global nos estarían hablando de una mejora en la selección de los parámetros de cada algortmo. En el caso de SVM (Figura 6) puede verse claramente la diferencia entre los parámetros por default y los mejores parámetros seleccionados. Siguiendo este análisis se consideran como mejores modelos los siguientes casos: KNN=3, Redes= 100, 50 y SVM 10, 1.

La siguiente tabla muestra el Precision, Recall y F1 promedio para las mejores performances de cada algoritmo. Puede verse que en todos los casos hay una mejora en la performance tanto a nivel facies individuales como global.

| | KNN | SVM | RED |
|---|---|---|---|
| Precision | 0.71 | 0.73 | 0.75 |
| Recall | 0.7 | 0.71 | 0.74 |
| F1 | 0.7 | 0.72 | 0.74 |

## Evaluando los clasificadores

El pozo Shankle well se separó al principio de este flujo de trabajo para evaluarlo como un nuevo dato. De él se tiene la misma información que antes (los mismos features con los que se entrenaron los modelos) y además, dado que ya es un pozo estudiado, también se conocen sus facies sedimentarias. Esto nos permite analizar la calidad de los modelos entrenados y la clasificación de las facies sedimentarias. En el código principal puede verse la clasificación con los modelos entrenados.

La tabla muestra los valores de las métricas promedio.

| | KNN | SVM | RED |
|---|---|---|---|
| Precision | 0.5 | 0.43 | 0.54 |
| Recall | 0.43 | 0.41 | 0.47 |
| F1 | 0.43 | 0.39 | 0.48 |

Y en las Figuras 12, 13 y 14 como son las facies ya estudiadas y las predichas por los distintos métodos.

Como puede verse la performance no es como uno hubiese esperado. Como se mencionó en el apartado anterior, siempre hay lugar para mejorar la selección de los parámetros para el algoritmo a utilizar. También debe de considerarse el tipo de dato con el que se cuenta. En este caso de facies geológicas, la performance puede mejorarse como se muestra en el tutorial (1) considerando que realmente las facies sedimentarias no son una unidad de roca completamente definidas por un único tipo de sedimento. Cada facie en general suele verse afectada por contenido de otros sedimentos en distintos porcentajes. Esta transición que puede existir entre ellas debería de considerarse de algún modo. Esto agrega una dificultad a los datos de entrada y procesamiento de los mismos, previo al entrenamiento de los clasificadores.

## Referencias:


1) https://github.com/seg/tutorials-2016/tree/master/1610_Facies_classification
2) Presentaciones y apuntes de clase
3) Hands-On machine learning with Scikit-Learn & Tensorflow