Práctica 1 - Aplicación del aprendizaje automático con el modelo SVM

December 29, 2022

Alumnos:

- Crespí Valero, Maribel
- Fortes Domínguez, Odilo

1 Introducción

En esta práctica el problema a resolver se basa en la creación de un modelo de aprendizaje automático que sea capaz de clasificar una palabra dada según pertenezca a la lengua inglesa o catalana.

Para ello se nos provee de un dataset con aproximadamente 1000 palabras en inglés y 1000 en catalán (son las mismas palabras). Tiene tres columnas, donde dos de ellas son la palabra en cada idioma, y la restante es el id de la fila.

El algoritmo a utilizar será un Support Vector Machine (SVM). SVM se basa en la idea de encontrar un hiperplano de separación entre dos clases de datos en un espacio de características de alta dimensión. Son muy efectivos en problemas de clasificación binaria y también pueden manejar problemas no lineales mediante técnicas de kernel.

2 Tratamiento del conjunto de datos

El tratamiento del conjunto de datos es una etapa muy importante a la hora de atacar cualquier problema. No podremos obtener buenos resultados si los datos tampoco son buenos.

Aunque los datos iniciales son los que son, podemos tratar de obtener nuevas características a partir de las ya existentes: seleccionar características relevantes, borrar características si vemos que no son relevantes, eliminar valores faltantes, corregir errores de datos y normalizar variables.

Un buen tratamiento del conjunto de datos puede mejorar significativamente la precisión y rendimiento de los modelos de aprendizaje automático utilizados posteriormente.

2.1 Preparación del conjunto de datos

Primero de todo deberemos echar un vistazo al dataset original.

[1]:
```
# Importamos la librería pandas para poder cargar y modificar el dataset.
import pandas as pd
```

```
# Cargamos el dataset.
df = pd.read_csv("data/data.csv", delimiter=r"\s+")
print(df)
```

```
        0      catala     angles
0       1         com         as
1       3        seva        his
2       4         que       that
3       5         ell         he
4       6         era        was
..    …          …        …
983   996         nas       nose
984   997      plural     plural
985   998      còlera      anger
986   999  reclamació      claim
987  1000   continent  continent

[988 rows x 3 columns]
```

Cabe mencionar que previamente hemos confirmado que todas las cadenas de texto están en minúscula, y además no hay valores nulos.

Ahora bien, como podemos observar, la primera columna contiene lo que parece ser un identificador de columna. No nos es útil, por lo que procedemos a eliminarla.

[2]:
```
# Borramos la columna que contiene el id de las filas.
df = df.drop('0', axis=1)
print(df)
```

```
         catala     angles
0           com         as
1          seva        his
2           que       that
3           ell         he
4           era        was
..          …        …
983         nas       nose
984      plural     plural
985      còlera      anger
986  reclamació      claim
987   continent  continent

[988 rows x 2 columns]
```

Como ya sabemos, para poder realizar un entrenamiento con un SVM debemos poder partir el dataframe en Datos y Target. En este caso no hay presente ningún Target (todavía).

Lo que haremos será transformar el dataframe para que haya una sola columna que contenga todas las palabras (de ambas lenguas), y una segunda columna que indique si es de un idioma o de otro.

[3]:
```
# Vamos a tratar por separado catalán e inglés y asignar el Target, luego los␣
 ↪combinaremos.
df2 = df

# Este tendrá sólo palabras catalanas y la columna Target contendrá el valor␣
 ↪'cat'.
# La columna catala la llamaremos 'Palabras' para luego unificar ambos␣
 ↪dataframes (las columnas deben tener el mismo nombre).
df = df.drop('angles', axis=1)
df = df.assign(Target=pd.Series(['cat'] * len(df)))
df = df.rename(columns={'catala': 'Palabras'})
print(df)
```

```
      Palabras Target
0          com    cat
1         seva    cat
2          que    cat
3          ell    cat
4          era    cat
..         …    …
983        nas    cat
984     plural    cat
985     còlera    cat
986  reclamació    cat
987   continent    cat

[988 rows x 2 columns]
```

[4]:
```
# Hacemos lo mismo pero con las palabras inglesas.
df2 = df2.drop('catala', axis=1)
df2 = df2.assign(Target=pd.Series(['eng'] * len(df2)))
df2 = df2.rename(columns={'angles': 'Palabras'})
print(df2)
```

```
     Palabras Target
0          as    eng
1         his    eng
2        that    eng
3          he    eng
4         was    eng
..         …    …
983      nose    eng
984    plural    eng
985     anger    eng
986     claim    eng
987  continent    eng

[988 rows x 2 columns]
```

[5]:
```
# Unimos los dataframes.
df = pd.concat([df, df2])
print(df)
```

```
      Palabras Target
0          com    cat
1         seva    cat
2          que    cat
3          ell    cat
4          era    cat
..         …    …
983       nose    eng
984     plural    eng
985      anger    eng
986      claim    eng
987  continent    eng

[1976 rows x 2 columns]
```

En teoría ya hemos acabado, pero se puede observar que aunque hay 1976 filas, la última tiene el id 987. Esto es porque se conserva el id automático previo a la fusión. Para solucionarlo resetearemos el índice.

[6]:
```
# Cada fila obtendrá su propio id único.
df = df.reset_index(level=0, drop=True)
print(df)
```

```
       Palabras Target
0           com    cat
1          seva    cat
2           que    cat
3           ell    cat
4           era    cat
…         …    …
1971       nose    eng
1972     plural    eng
1973      anger    eng
1974      claim    eng
1975  continent    eng

[1976 rows x 2 columns]
```

2.2 Selección de características

Llegados a este punto ya tenemos una estructura más acorde con lo que se necesita para realizar un entrenamiento.

Sin embargo sólo tenemos una única característica, por lo que el resultado no será para nada bueno.

Vamos a tratar de añadir características relevantes para proporcionar información adicional que el

modelo pueda utilizar para tomar decisiones.

- Longitud de la palabra
- Número de vocales
- Contiene tilde
- Contiene combinaciones de letras exclusivas
- Frecuencia de cada vocal

Para algunos casos hemos creado ciertas funciones. Iremos mostrando uno a uno cada caso de adición de características.

Longitud de la palabra: Simplemente aplicaremos la función len.

[7]:
```
df['Num_letras'] = df['Palabras'].apply(len)
print(df)
```

```
      Palabras Target  Num_letras
0          com    cat           3
1         seva    cat           4
2          que    cat           3
3          ell    cat           3
4          era    cat           3
…          …    …           …
1971      nose    eng           4
1972    plural    eng           6
1973     anger    eng           5
1974     claim    eng           5
1975  continent    eng           9

[1976 rows x 3 columns]
```

Número de vocales: Creamos una función básica que cuente las vocales de una palabra.

[8]:
```
def contar_vocales(palabra):
    vocales = ['a', 'e', 'i', 'o', 'u']
    contador = 0
    for letra in palabra:
        if letra in vocales:
            contador += 1
    return contador

df['Num_vocales'] = df['Palabras'].apply(contar_vocales)
print(df)
```

```
      Palabras Target  Num_letras  Num_vocales
0          com    cat           3            1
1         seva    cat           4            2
```

```
2           que    cat           3            2
3           ell    cat           3            1
4           era    cat           3            2
…         …    …         …          …
1971       nose    eng           4            2
1972     plural    eng           6            2
1973      anger    eng           5            2
1974      claim    eng           5            2
1975  continent    eng           9            3

[1976 rows x 4 columns]
```

Contiene tilde: La función creada comprueba todos los posibles carácteres existentes que puedan llevar una tilde o similar (por ejemplo ü). Esta característica nos puede beneficiar ya que las palabras en inglés nunca llevan tilde.

[9]:
```
def contiene_tilde(palabra):

    tildes = "áéíóúÁÉÍÓÚàèìòùÀÈÌÒÙüÜ"
    for letra in palabra:
        if letra in tildes:
            return 1
    return 0

df['Tilde'] = df['Palabras'].apply(contiene_tilde)
print(df)
```

```
       Palabras Target  Num_letras  Num_vocales  Tilde
0           com    cat           3            1      0
1          seva    cat           4            2      0
2           que    cat           3            2      0
3           ell    cat           3            1      0
4           era    cat           3            2      0
…         …    …         …          …    …
1971       nose    eng           4            2      0
1972     plural    eng           6            2      0
1973      anger    eng           5            2      0
1974      claim    eng           5            2      0
1975  continent    eng           9            3      0

[1976 rows x 5 columns]
```

Contiene combinaciones de letras exclusivas: No se puede ignorar que algunas combinaciones son mucho más recurrentes en una lengua que en otra. Por ejemplo 'th'. Concretamente añadiremos 3 combinaciones con las que sabemos que ocurre esto.

[10]:
```
def contiene_combinacion(combinacion,palabra):
    if combinacion in palabra:
```

```
        return 1
    else:
        return 0

df['Contiene_qu'] = df['Palabras'].apply(lambda x:␣
 ↪contiene_combinacion(combinacion="qu", palabra=x))
df['Contiene_th'] = df['Palabras'].apply(lambda x:␣
 ↪contiene_combinacion(combinacion="th", palabra=x))
df['Contiene_ll'] = df['Palabras'].apply(lambda x:␣
 ↪contiene_combinacion(combinacion="ll", palabra=x))
print(df)
```

```
       Palabras Target  Num_letras  Num_vocales  Tilde  Contiene_qu  \
0           com    cat           3            1      0            0
1          seva    cat           4            2      0            0
2           que    cat           3            2      0            1
3           ell    cat           3            1      0            0
4           era    cat           3            2      0            0
…           …    …         …          …    …          …
1971       nose    eng           4            2      0            0
1972     plural    eng           6            2      0            0
1973      anger    eng           5            2      0            0
1974      claim    eng           5            2      0            0
1975  continent    eng           9            3      0            0

      Contiene_th  Contiene_ll
0               0            0
1               0            0
2               0            0
3               0            1
4               0            0
…           …          …
1971            0            0
1972            0            0
1973            0            0
1974            0            0
1975            0            0

[1976 rows x 8 columns]
```

Frecuencia de cada vocal: Esta característica juega un papel muy importante (y lo sabemos porque sin ella los resultados eran bastante peores).

Para efectuarla, primero cambiaremos todas las vocales con tilde por vocales sin ella y posteriormente tan sólo contaremos vocales. Crearemos una columna por vocal.

[11]:
```
def quitar_tildes(palabra):
    cambios = {'a': ['à', 'á'],
               'e': ['è', 'é'],
               'i': ['ì', 'í', 'ï'],
               'o': ['ò', 'ó'],
               'u': ['ù', 'ú', 'ü']
                }
    for clave, valores in cambios.items():
        for valor in valores:
            palabra = palabra.replace(valor, clave)

    return palabra

def contar_todas_las_vocales(palabra):
    palabra = quitar_tildes(palabra)

    vocales = "aeiou"
    frecuencia_vocales = [0, 0, 0, 0, 0]  # inicializamos el vector con ceros

    # recorremos cada caracter de la palabra
    for caracter in palabra:
        # si el caracter es una vocal, aumentamos el contador correspondiente
        if caracter in vocales:
            indice = vocales.index(caracter)
            frecuencia_vocales[indice] += 1

    return frecuencia_vocales

df[["A", "E", "I", "O", "U"]] = pd.DataFrame(df["Palabras"].
 ↪apply(contar_todas_las_vocales).tolist(), index=df.index)
print(df)
```

```
       Palabras Target  Num_letras  Num_vocales  Tilde  Contiene_qu  \
0           com    cat           3            1      0            0
1          seva    cat           4            2      0            0
2           que    cat           3            2      0            1
3           ell    cat           3            1      0            0
4           era    cat           3            2      0            0
…           …      …             …            …      …            …
1971       nose    eng           4            2      0            0
1972     plural    eng           6            2      0            0
1973      anger    eng           5            2      0            0
1974      claim    eng           5            2      0            0
1975  continent    eng           9            3      0            0

      Contiene_th  Contiene_ll  A  E  I  O  U
0               0            0  0  0  0  1  0
1               0            0  1  1  0  0  0
```

```
2                0              0  0  1  0  0  1
3                0              1  0  1  0  0  0
4                0              0  1  1  0  0  0
…              …            … .. .. .. .. ..
1971             0              0  0  1  0  1  0
1972             0              0  1  0  0  0  1
1973             0              0  1  1  0  0  0
1974             0              0  1  0  1  0  0
1975             0              0  0  1  1  1  0

[1976 rows x 13 columns]
```

3 Selección de métricas

Cuando más tarde obtengamos nuestro modelo, podremos generar un reporte que nos informa del accuracy, precision, recall y f1-score gracias a la librería classification_report de sci-kit learn.

[12]:
```
from sklearn.metrics import classification_report
```

Nosotros nos basaremos en el accuracy para evaluar y escoger el mejor modelo (cuanto más alto este valor, mejor).

4 Entrenamiento del modelo

Una vez ya tenemos un buen dataframe de datos con el que trabajar y una métrica seleccionada, es hora de realizar el entrenamiento del modelo.

Primeramente vamos a dividir el dataframe: por un lado en las características a aprender, y por el otro lado el Target.

Importante: 'Palabras'. La única característica original del dataset es un String. Esto nos dificulta el entrenamiento, ya que la función fit del entrenamiento no acepta cadenas de texto. Para ello hay que vectorizar estos Strings, pero no vale la pena ya que el uso de esta característica no contribuye a ningún tipo de mejora sobre el accuracy del modelo.

Por este motivo, no añadiremos la característica 'Palabras' a ningún conjunto.

[13]:
```
X = df.loc[:, ['Num_letras', 'Num_vocales', 'Tilde','Contiene_qu',␣
 ↪'Contiene_th', 'Contiene_ll', 'A', 'E', 'I', 'O', 'U']]
y = df['Target']
```

4.1 Conjuntos de entrenamiento y test

Hemos elegido que el 30% de los datos conformen el conjunto de test. Queremos asegurarnos de que el modelo es evaluado correctamente, y aún así nos quedarán bastantes datos para un entrenamiento adecuado.

Fijaremos una seed para que el proceso nos dé siempre los mismos resultados.

[14]:
```
from sklearn.model_selection import train_test_split

# Establecemos el tamaño del conjunto de test y fijamos la seed.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 6)
```

Aunque el porcentaje del conjunto de test en cuanto a los datos totales es relativamente arbitrario (siempre suele rondar el 20-30%), existen ciertos hiperparámetros de SVM que dan lugar a un gran número de diferentes combinaciones, demasiadas si quisiéramos probarlas todas una a una.

Para obtener la mejor combinación de hiperparámetros que nos dé el mayor accuracy y por tanto el mejor modelo posible (según nuestras métricas), combinaremos dos técnicas que nos permitirán obtener los valores óptimos de los hiperparámetros: K-Fold y Grid Search.

4.2 Combinación de K-Fold con Grid Search

También conocido como Nested Cross-Validation, este algoritmo se basa en dos bucles anidados: el externo divide el conjunto de entrenamiento mediante K-Folding, y en el interno se realiza la búsqueda de los mejores hiperparámetros.

Cabe resaltar que esta búsqueda no es absoluta, pues está limitada a los hiperparámetros que nosotros proponemos.

Los hiperparámetros que propondremos serán:

- Kernel: Lineal y Radial

Realmente hemos probado también otros dos: Polinomial y Sigmoide, pero si se ponen los 4 juntos entonces el tiempo de ejecución se vuelve excesivamente largo. Para que sea fácil probar el programa entero sólo indicamos estos dos.

- C: 1, 10, 100

C controla la cantidad de tolerancia al error en la clasificación. Cuanto más alto el valor de C menor tolerancia al error y por lo tanto se obtiene un modelo más complejo, pues se ajusta más a los datos de entrenamiento, aunque también puede llevar a un sobreajuste del modelo. Para valores bajos, se ignoran ciertas muestras para dar mayor libertad al hiperplano solución.

- Gamma: 0.1, 1, 10

Gamma es un hiperparámetro propio del kernel Radial (RBF), y controla la distancia de influencia de una muestra. Para valores altos las muestras deben estar muy cerca para que se considere que pertenecen a la misma clase. Para valores bajos se es más permisivo, por lo que habrá más puntos agrupados con mayores distancias.

[15]:
```
param_grid = {
    'kernel': ['linear', 'rbf'],
    'C': [1, 10, 100],
    'gamma': [0.1, 1, 10],
}
```

Ahora crearemos nuestro objeto clasificador con la función GridSearchCV a la que pasamos varios parámetros:

- estimator

Aquí indicamos el algoritmo a utilizar, que es SVM. Pero deberemos poner SVC ya que es la implementación específica de SVM para problemas de clasificación binaria en scikit-learn.

- param_grid

Como dijimos, param_grid contiene los valores de hiperparámetros que vamos a probar.

- cv

Se usará la Cross-Validation por defecto, aunque si bien podríamos haber definido nosotros mismos un número exacto de divisiones para realizarla.

- verbose

Indica si queremos obtener información a medida que se realiza el GridSearch. Como no queremos, lo ponemos a 0.

- scoring

Indica qué métrica usará el GridSearch para juzgar si un modelo es mejor que otro. Como dijimos anteriormente, nosotros usaremos el accuracy.

[16]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(estimator=SVC(), param_grid=param_grid, cv=None, verbose=0,␣
 ↪scoring='accuracy')
```

Ahora por fin realizaremos el entrenamiento. Los resultados los revisaremos en la siguiente sección.

[17]:
```
clf.fit(X_train, y_train)
```

[17]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10, 100], 'gamma': [0.1, 1, 10],
                         'kernel': ['linear', 'rbf']},
             scoring='accuracy')
```

5 Resultados

[18]:
```
print(clf.best_params_)
print(clf.best_score_)
```

```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.7035368597289804
```

Como podemos observar, el kernel que nos ha dado el mejor resultado (mayor accuracy) ha sido el Radial (rbf). Los hiperparámetros correspondientes son C=1 y gamma=0.1, por lo que se entiende que hay mucha tolerancia a errores (C) y que los puntos pertenecientes a una misma clase no están tan juntos como se podría (gamma).

El accuracy ha sido del 70%, o lo que es lo mismo, 7 de cada 10 predicciones hechas por el modelo son correctas.

Recordemos que el objetivo final era obtener un modelo que clasificara lo mejor posible una palabra dada según fuera de la lengua inglesa o catalana. Aquí lo tenemos:

[19]:
```
biel_moya = clf.best_estimator_
```

Vamos a obtener los resultados finales con el conjunto de test.

[20]:
```
# Obtenemos las predicciones de X_test.
y_pred = biel_moya.predict(X_test)

# Comparando las predicciones con el ground truth podremos ver
# qué tan bien lo hace y revisar diferentes métricas.
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

         cat       0.76      0.66      0.71       288
         eng       0.72      0.81      0.76       305

    accuracy                           0.74       593
   macro avg       0.74      0.73      0.73       593
weighted avg       0.74      0.74      0.73       593
```

A partir del informe de clasificación proporcionado, se pueden sacar las siguientes conclusiones:

- El modelo tiene un accuracy de 0.74 en general, lo que significa que el modelo es correcto en un 74% de las veces.

Sin embargo previamente habíamos visto que 'clf.best_score_' nos daba un accuracy de 0.703 (recordemos que habíamos indicado que la métrica usada para elegir fue el accuracy).

Esto se debe a que GridSearchCV y classification_report usan métodos diferentes para calcular el accuracy.

En el primer caso, el accuracy se calcula como el número de predicciones correctas dividido entre el número total de predicciones, mientras que classification_report se basa en la matriz de confusión del modelo, y no sólo en el número total de predicciones correctas.

- La precisión para la clase "cat" es de 0.76 y para la clase "eng" es de 0.72, lo que significa que el modelo es correcto en un 76% de las veces en las que se le da una muestra etiquetada como "cat", y un 72% de las veces en las que se le da una muestra etiquetada como "eng".
- El recall para la clase "cat" es de 0.66, lo que significa que el modelo detecta el 66% de las muestras etiquetadas como "cat". En el caso de la clase "eng" el recall es de 0.81, por lo que el modelo detecta el 81% de las muestras etiquetadas como "eng".
- El puntaje F1 para ambas clases es mayor de 0.7. Esto significa que el modelo tiene una buena precisión y un buen recall para ambas clases.
- El modelo tiene un soporte total de 593 muestras (X_test), con 288 muestras etiquetadas como "cat" y 305 muestras etiquetadas como "eng".

6 Conclusión

Los resultados mostraron que el SVM tuvo una accuracy del 74% en la clasificación de las muestras del conjunto de datos de test.

En general, se puede concluir que el SVM es una herramienta efectiva para resolver problemas de clasificación y es capaz de manejar conjuntos de datos con muchas características.

Además, SVM tiene la ventaja de ser un modelo no paramétrico, lo que significa que no requiere la especificación de una función de forma explícita y puede adaptarse a diferentes formas de distribución de los datos.

Sin embargo, es importante tener en cuenta que SVM puede ser sensible a la elección del kernel e hiperparámetros, y puede requerir un ajuste cuidadoso de éstos para obtener buenos resultados (de ahí la decisión de utilizar Grid Search).

En resumen, SVM es una gran opción a considerar en problemas de clasificación.

Finalmente de forma ajena al problema, podemos afirmar que esta práctica nos ha ayudado enormemente a aprender y reforzar los conocimientos obtenidos en esta asignatura.