PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
IIC2613 - INTELIGENCIA ARTIFICIAL

# TAREA 3

**Fecha Máxima de Entrega: 28/06**

# Objetivo y Regla importante

En esta tarea veremos el uso práctico de tres técnicas o estrategias clásicas: SVM, clasificador o regresor bayesiano ingenuo, y árboles de decisión. Siendo éste nuestro primer acercamiento a la lógica inductiva en inteliegencia artificial, será importante revisar algunas herramientas que son de común uso en esta área.

Antes de seguir avanzando, y como ha sido costumbre en el curso vamos a repasar nuestra regla de integridad, con una actualización para esta tarea. Estamos conscientes que algunas respuestas a las preguntas de esta tarea se pueden encontrar en Internet, en libros, o en conversaciones con tus compañeros, de hecho esta tarea tiene un ítem de investigación. Como esta tarea tiene un fin pedagógico, la regla de integridad académica que pedimos que cumplas es la siguiente:

*Toda respuesta a una pregunta teórica debe ser escrita por ti, individualmente. Esto significa que al momento de editar la respuesta, no debes copiar el material hecho por otra persona.* ***Queremos tu comprensión y entendimiento de las cosas que estudies y aprendas****. Por lo tanto, las definiciones y análisis que presentes deben venir de tu propio razonamiento.*
*También es importante dejar claro que sabemos que estás utilizando material hecho por otras personas, y debes reconocerlo.* ***Es por eso que te pediremos que cites tus fuentes en esta oportunidad****. Tienes dos formas de citar. La primera es bastante indirecta, y solamente exige que menciones a tu fuente en tu bibliografía. Harás esto cuando hayas aprendido conceptos, los hayas interiorizado, y estés dando tu interpretación. La segunda es más directa, utilizarás las comillas "" para parafrasear algo que se le ocurrió a otra persona, seguido de un paréntesis con la referencia de dónde se encuentra dicho texto. Solamente debes parafrasear cuando quieres darle fuerza a un argumento, jamás para contestar algo que se te pregunte a ti, pues tal respuesta, si bien te evita el plagio, tampoco te da puntaje. Las citas de publicaciones científicas las puedes poner en cualquier formato de amplia aceptación como APA, ICONTEC e IEEE, por nombrar algunos. Las citas a sitios web las debes hacer referenciando el link del sitio, y la fecha de la que has tomado la referencia. Por favor, no cites a tus compañeros, aprende de ellos, sé crítico con sus opiniones, y entrega tu propio trabajo original.*
*Esta misma regla también se cumple respecto del código que entregues.*

Tu entrega deberá incluir un archivo llamado INTEGRIDAD.txt, en donde afirmas que has leído, entendido y respetado la regla de arriba. De no cumplirse esto, **tu tarea no será corregida, y en consecuencia obtendrás la nota mínima, tal como si no hubieses entregado**. Recuerda subir todo el material relacionado a tu tarea en tu repositorio personal de GitHub a más tardar la fecha de entrega a las 23:59, para esto deben crear el directorio `T3/` que contenga todos los archivos y subdirectorios especificados.

## 1 Librerías

La actividad la realizaremos utilizando códigos en lenguaje de programación Python. Utilizaremos diversas librerías de este lenguaje tanto para cargar, como para preprocesar los datos, y clasificarlos. Prácticamente cada etapa de este tratamiento de datos está liderado por una de estas librerías, cada una de las cuales merece

su apropiada atención.

## 1.1 Pandas

Es bastante común encontrar sets de datos estructurados en formatos de tablas. La herramienta *Pandas* es especialmente útil para manejar este tipo de datos. Como librería, es una extensión de *NumPy*, pero sus mayores atributos están en la forma en la que se indexan los datos. La clase DataFrame permite hacer un manejo bastante cómodo de los datos. En general, se denominará DataFrame a la tabla de datos cuando éstos ya estén cargados, y Series a las columnas.
Para importar la librería simplemente utilizamos la palabra reservada `import`. Por convención, utilizamos el diminutivo `pd`:

```
import pandas as pd
```

Por convención un DataFrame se denomina `df`. Muchos de ellos vienen guardados en formato *.csv*. Para cargarlos se puede hacer:

```
df = pd.read_csv('ruta al archivo')
```

El DataFrame puede ser cargado por su nombre, pero es preferible ver un resumen de su estructura con el método `head`. El llamado a este método desplegará los primeros cinco elementos (filas), lo que es más rápido que un despliegue directo.

De manera similar a la librería *NumPy*, el número de filas y columnas se puede conocer con el atributo `shape`. Por otro lado también pueden resultar de utilidad los métodos `info` y `describe`. Mientras el primero devuelve características, número de columnas, cuales son, y el número de elementos no nulos, el segundo proporciona información básica estadística del set de datos.

Si queremos limipar el DataFrame de datos indeseados deberemos conocerlos primero. Para ello se puede utilizar el método `isnull`. Los elementos nulos aparecen con el denominativo **NaN**, y el método `isnull` devolverá un booleano con un valor corresponiente a si se encuentra dicho denominativo o no. De la misma manera el método `duplicated` nos ayudará a encontrar duplicados.

Navegar sobre los datos es relativamente intuitivo si se conoce la librería *NumPy*. Una columna o serie puede ser accedida como si fuese un atributo de valor `name`, como por ejemplo `df.name`, o como si fuese un diccionario con una llave: `df['name']`. Para acceder a dos series se utiliza la extensión `df[['name1'], ['name2']]`. Una lista de todas las series disponibles se obtiene con `df.columns`. Finalmente, para acceder a un elemento dentro de una columna se debe aplicar su índice luego de haber llamado a la serie. Por ejemplo al elemento 354 de `name` se accede haciendo `df['name'][354]`.

Una vez hemos podido acceder a ciertas series, o a elementos de ellas, es posible trabajar de manera numérica con cierta facilidad. Algunas operaciones matemáticas son bastante simples y directas, mientras que otras pueden requerir un poco más de inmersión en la docuentación de la libreía.

```
q = df['name1'][354] + df['name2'][726]
#esta es una operacion simple sobre dos celdas del DataFrame
m = df['name3'].mean()
s = df['name4'].sum()
#estas son operaciones sobre toda la serie
```

Una operación numérica interesante sobre la serie es la de contar `value_counts` que me retornará la cantidad de elementos de cada clase de una serie. En particular, las clases de esa serie se pueden recuperar con el método `unique`.

Existen accesos más avanzados que se pueden invocar con el método `loc` o el método `groupby`. Estos métodos permiten trabajar con grandes cantidades de datos, hacer consultas y agrupaciones. El método `loc` permite filtrar filas o columnas por medio una etiqueta o un booleano. Un ejemplo puede ser:

```
df.loc[df['name1'] > 6, ['name2']]
```

En este ejemplo se filtrarán todas las filas, cuya serie de nombre `name1` tengan un valor mayor que 6, pero se recuperarán los valores de la serie `name2` de esas filas.

Por otro lado, el método `groupby` retornará un objeto con varias propiedades interesantes. Típicamente se utiliza sobre valores no numéricos a los que pueden ser aplicadas otras operaciones o métodos:

```
df.groupby(['name1']).sum()
```

Si la serie `name1` tiene 5 instancias, el método mostrará una tabla de 5 filas para las que intentará sumar sus valores en las otras series. Otra aplicación puede requerir iterar sobre ese objeto:

```
for (label, group) in df.groupby(['name1']):
    new_group = group.name2 + group.name3
```

En esta otra aplicación de ejemplo se obtiene una nueva serie, para cada grupo, que resulta de la adición de otras dos series. La variable `label` contiene uno de los posibles valores que `name1` puede tomar. Por otra parte `group` es un DataFrame que solamente contiene el valor actual que está tomando `label` en la serie `name1`.

En ambos casos, un valor no numérico que el método `groupby` podría tomar con más facilidad que otros métodos, podría ser, por ejemplo, la serie `mascotas`, cuyas instancias serían para este ejemplo [`perros`, `gatos`, `hamsters`].

Finalmente, si lo que se desea es guardar un DataFrame procesado se puede usar el método `to_csv`:

```
df.to_csv('nombre del archivo csv')
```

## 1.2 Scikit-learn

Adicionalmente, usaremos la librería scikit-learn `http:\\www.scikit-learn.org`, la cual provee una gran gama de técnicas de aprendizaje de máquina, tal como clasificadores del tipo SVM. La orientación de la librería es hacia objetos, aunque también importaremos algunas funciones útiles. Primero que todo, para esta oportunidad nos interesan los clasificadores que podemos importar haciendo:

```
from sklearn.svm import SVC
#Clase que utilizaremos para un clasificador SVM
from sklearn.naive_bayes import GaussianNB
#Clase que utilizaremos para un clasificador Bayesiano ingenuo
from sklearn.tree import DecisionTreeClassifier
#Clase que utilizaremos para un clasificador de arbol de decision
```

A la hora de trabajar con estas clases se debe tener en cuenta lo siguiente:

- Todos los clasificadores pueden ajustar sus hiperparámetros con el método `set_params`.
- El método `fit(X,y)` ejecutará el procedimiento de optimización de parámetros o entrenamiento del clasificador.
- El método `predict(X)` generará un vector $\hat{y}$ con la predicción del clasificador para la matriz $X$.

## 1.3 Dlib

La librería *dlib* fue creada con la intención de ser un framework relativamente simple de utilizar para detección y reconocimiento de rostros. Si bien es una herramienta relativamente antigua se pueden hacer cosas bastante interesantes con esta librería. Se puede instalar con el comando:

```
pip3 install face_recognition
```

Al importar la librería simplemente se debe hacer:

```
import face_recognition as fr
```

Su uso consta de algunas funciones que requieren del uso de una GPU. Más abajo veremos cómo tener un acceso gratis y temporal a una. Por el momento estos son algunos comandos útiles:

- `image = fr.load_image_file("my_picture.jpg")`: Sirve para tener un objeto imagen en un formato útil para la librería.
- `face_locations = fr.face_locations(image, model="cnn")`: Genera un arreglo de varios puntos que representan cajas contenedoras donde se han hallado rostros.
- `face_landmarks_list = fr.face_landmarks(image)`: Entrega un diccionario con varias listas de puntos que se se corresponden con marcas faciales. Un ejemplo de un modelo de 68 puntos se puede ver en la Figura 1.
- `my_face_encoding = fr.face_encodings(picture_of_me)[0]`: Entrega un vector con una codificación que representa a un rostro.
- `results = fr.compare_faces([my_face_encoding], unknown_face_encoding)`: Devuelve un booleano si ha podido establecer una similitud suficiente entre `my_face_encoding` y `unknown_face_encoding` para determinar si un rostro se parece o no a otro.


Figure 1: Modelo de puntos de dlib

## 2 Google Colab

Arquitecturas como ésta son posibles gracias a las GPUs. Estos procesadores se han desarrollado debido al incentivo económico que la industria de videojuegos ha generado. Ahora, han demostrado una gran utilidad en el campo del aprendizaje automático por su alto poder de procesamiento paralelo. Sin embargo son procesadores costosos y requieren de drivers específicos para manejarlos. Felizmente Google ha dispuesto una herramienta gratuita para poder aprovechar su uso, la herramienta Google Colab `https://colab.research.google.com/`.

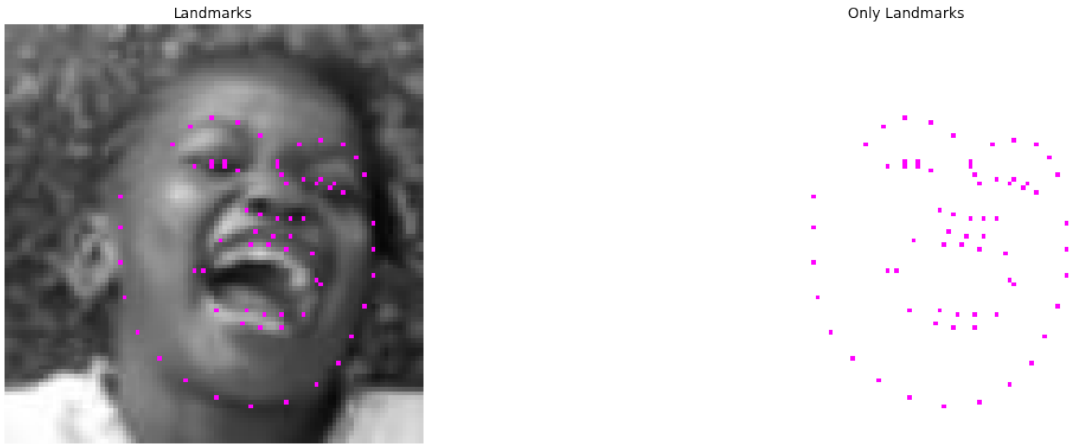


Figure 2: Puntos aplicados a un rostro del set de datos.

# 3 Métricas de rendimiento

Para conocer el rendimiento de un modelo ya entrenado, es necesario probar su capacidad para predecir nuevos datos (ie. datos no usados en su entrenamiento). Ya que la cantidad de datos de la que se dispone es limitada, es necesario separar de antemano una porción de los datos (set de test) que se usará para obtener nuestras métricas.

## 3.1 Métrica de evaluación

Una de las métricas más simples para esta tarea es el puntaje de precisión (accuracy score). Este puntaje se obtiene comparando las respuestas correctas (datos clasificados en la categoría correcta) con el total de las predicciones hechas.

$$accuracy\_score = \frac{\#Predicciones\_correctas}{\#Predicciones\_totales}$$

La fórmula anterior puede ser multiplicada por 100 para obtener el porcentaje de acierto sobre el set de test.

```
from sklearn.metrics import accuracy_score
#acc = accuracy_score(y_true, y_pred)
```

## 3.2 Matriz de confusión

Si bien el puntaje de precisión nos da una vista preliminar al rendimiento de nuestro modelo, nos interesa saber en qué casos este se equivoca o qué tan bueno es para predecir ciertas categorías específicas. Para esto otra métrica interesante son las matrices de confusión. Estas también se construyen comparando las predicciones con las categorías reales del set de test. Cada celda de la matriz contiene la cantidad de datos correspondiente a la clase indicada con el rótulo o índice fila que fueron erróneamente clasificados como parte de la clase indicada con el rótulo o índice columna. Por ejemplo, en la figura 2 para un clasificador de 7 categorías (clases 0 a 6), se puede observar que 19 ejemplos de la **clase 3**, fueron clasificados erróneamente como **clase 5** (*Nota: en algunos textos la matriz de confusion puede ser definida según una convención inversa, es decir, columnas indicar la clase correcta y filas las predicciones.*).
Con esto se deduce que la diagonal de la matriz representa los datos del set que fueron correctamente predichos.
Una práctica común a la hora de observar matrices de confusión es obtener una matriz normalizada con el total de los datos. Con esto cada celda representa la porción de datos clasificados en dicha categoría contra la respuesta correcta.

Predicción

Real

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **0** | 972 | 1 | 1 | 0 | 0 | 1 | 3 |
| **1** | 0 | 1129 | 3 | 0 | 1 | 1 | 1 |
| **2** | 7 | 6 | 992 | 5 | 1 | 0 | 2 |
| **3** | 0 | 1 | 2 | 970 | 1 | 19 | 0 |
| **4** | 0 | 7 | 0 | 0 | 944 | 0 | 3 |
| **5** | 1 | 1 | 0 | 12 | 2 | 860 | 5 |
| **6** | 4 | 2 | 0 | 0 | 3 | 5 | 944 |

Figure 3: Ejemplo de matriz de confusión para 5 categorías

```
from sklearn.metrics import confusion_matrix
#cm = confusion_matrix(y_true, y_pred)
```

# 4 Actividades

## 4.1 Problema de clasificación

El objetivo de esta actividad es clasificar emociones en rostros. Se le proporcionará un set de datos con imágenes de rostros, y usted deberá poder identificar la emoción mediante las técnicas aprendidas en clases. Para ello puede elegir un clasificador, bien sea SVM, Bayesiano ingenuo, o Árbol de Decisión, como estrategia inductiva.

Prepare un *jupyter notebook* con las respuestas. Descargue el set de datos utilizando la siguiente dirección de *google drive* https://drive.google.com/file/d/1rR0CjYZDgMfZDVYlyPxJdYpDN1XH5AYq. Puede hacer uso de la instrucción *gdown –id [id]* donde el [id] se reemplaza por lo que está marcado en rojo. Esta instrucción importa los datos a la carpeta en la que se encuentra el archivo *jupyter*.

Este set de datos contiene imágenes en blanco y negro de rostros de personas en un tamaño de 48x48. Cada rostro está demostrando una emoción básica de este grupo de categorías: [*enojo*, *asco*, *miedo*, *alergría*, *triteza*, *sorpresa*, *neutral*].

Los datos se encuentran en un archivo *.csv*, para lo que será necesario el uso de la librería *Pandas*. Cargue el archivo en una instancia de la clase *Dataframe* de esta librería y despliegue un resumen utilizando la instrucción `head`. Como podrá comprobar, el archivo contiene tres columnas: **emoción**, **pixeles**, **uso**. La primera es la categoría a la que pertenece la emoción en la imagen, que corresponde al índice de la lista de emociones de arriba. La segunda columna son los pixeles de la imagen, escritos como un número de 0 a 255, y separados por un espacio en blanco. La tercera columna presenta el uso que tiene ese dato (fila) en el set. En principio, cada dato ya tendría asignado su rol, como de entrenamiento, validación o prueba. En vez de eso, hemos separado el set en dos, uno como “Test”, y el otro como “ToV”. Utilizaremos el primer set para una prueba generalizada de los resultados de la tarea, mientras que quedará bajo su criterio el hacer una separación adecuada entre los datos de entrenamiento y validación. Como podrá conjeturar el set de datos tiene miles de filas, cada una correspondiente a una muestra.

1. Haga una buena lectura de los datos contenidos en el archivo csv. Para ello identifique y transforme los datos a un formato adecuado para su trabajo en esta actividad. Despliegue una imagen de cada clase

del set de datos. Junto a la imagen debe señalar la emoción representada en forma clara, (simplemente despliegue la emoción correspondiente en texto). Utilice *dlib* para obtener la nube de puntos del rostro con la función `face_landmarks`. Vuelva a desplegar las mismas imágenes ahora utilizando el formato que tiene la Figura 2. Investigue sobre las librerías disponibles para el trabajo de imágenes en *Python*.

2. Separe el set de datos según su uso en entrenamiento, validación y test, de acuerdo a las instrucciones dadas anteriormente. Explique y justifique su estrategia para la división de las filas con valor **uso** de "ToV". Construya las matrices de caratacterísticas $X$, y los vectores de etiquetas $y$ correspondientes a cada set. ¿Cuántas muestras tiene cada set? ¿Cuántos ejemplos hay por cada clase en cada set? ¿Cuánto es el máximo de características que puede incluir en cada matriz $X$?

3. Entrene todos los clasificadores. Para los clasificadores SVM y árbol de decisión haga un estudio de hiperparámetros, utlizando el método `set_params`. Con SVM seleccione tres valores de *kernel* y cinco valores de *slack*. Con el árbol tome los dos valores posibles de *criterion* y seleccione siete valores para *max_depth*. El clasificador bayesiano ingenuo pide como hiperparámetros los *priors* del problema, que no poseemos, por lo que solo entrenaremos con este método una vez sin variar hiperparámetros. Una vez haya terminado reporte la exactitud y la matriz de confusión con el mejor set de hiperparámetros de cada método. ¿Qué clasificador tiene mejor resultado con este problema? ¿Cuál es la posible razón? ¿Por qué le podría ir mal al peor de los clasificadores? Justifique cada pregunta consideranto tanto la materia como las características propias de este problema.

4. Analice los resultados de la matriz de confusión del mejor clasificador con el mejor set de hiperparámetros seleccionado. ¿Cuál es el error más común?, ¿Qué categoría obtiene el mejor rendimiento? ¿Cuál es el más bajo? Indique razones que puedan justificar estos resultados. Para ello mire las imágenes reales incluidas en el set de datos.

5. Seleccione el mejor clasificador evaluado por usted utilizando el mejor set de hiperparámetros encontrados. Evalúelo, en el set de test. Reporte tanto la exactitud como la matriz de confusión. ¿Cómo cambian los resultados con respecto a lo obtenido en la parte 4? Haga un análisis comparativo entre los resultados aquí obtenidos y los de la parte 4 considerando las preguntas hechas en 4.

6. Proponga una arquitectura de ensamblaje de clasificadores utilizando únicamente SVM, Árboes de decisión y/o Bayes Ingenuo. Justifique la arquitectura en base a su conocimiento obtenido en clases, su experiencia con el set de datos, y las referencias que pueda encontrar en línea. Sea cuidadoso(a) de citar dichas referencia cuando corresponda. Como mínimo entregue un diagrama de bloques que represente la arquitectura diseñada. Finalmente, haga una investigación sobre las técnicas de *Bagging* y *Boosting*. Defina ambas técnicas y haga un análisis comparativo de ellas. ¿Cuál de ellas elegiría para este set de datos? ¿Por qué? Justifique.

El entregable es un archivo *jupyter notebook* (de extensión ".ipynb") con su nombre y apellido, número de alumno, y el desarrollo de la actividad. Cerciórese de que la opción "Omitir el resultado de las celdas al guardar este notebook" está deshabilitada y que la aceleración de hardware está configurada en "GPU". Puede hacer esto en el menú Edición⇒ Configuración del notebook. No se corregirán los notebooks que no tengan los resultados disponibles.