CS2601
INTELIGENCIA ARTIFICIAL

Proyecto 2: Clasificación de emociones en rostros

Mauricio Pinto
Francesco Uccelli
Juan Manuel Navarro

30 de Mayo del 2021

1 Introducción

La clasificación de imágenes se refiere a un proceso en la visión por computadora que puede clasificar una imagen de acuerdo con su contenido visual. Por ejemplo, se puede diseñar un algoritmo de clasificación de imágenes para decir si una imagen contiene una figura humana o no. El concepto de clasificación de imágenes nos ayuda a diferenciar un conjunto de datos e identificar cada imagen. La clasificación de imágenes es una de las aplicaciones más populares de la visión por computadora y un concepto imprescindible para cualquiera que desee desempeñar un papel en este campo. En este informe, se detallara los algoritmos utilizados sobre una base de datos para poder clasificar imágenes en base a las emociones faciales que muestren.

1.1 Vectores característicos

Para el cálculo de los vectores característicos de las imágenes con las distintas emociones usamos la librería PyWavelets. Esta librería utiliza un método de compresión con wavelet que transforma la imagen en un conjunto de coeficientes que luego se deben aplanar con la función a continuación para introducirla en los modelos.

```
    def flatten(self, matrixs):
        vectors=[]
        for i in range(len(matrixs)):
            vectors = vectors +
            [matrixs[i].reshape(24*24).tolist()]
        return vectors
```

Por otro lado para generar los datos con los diferentes folders de emociones usamos:

```
    def generate_datasets (self):
        self.x = []
        self.y = []
        i = 0
        for path in self.paths:
            features = []
            for image_path in os.listdir(path):
                input_path=os.path.join(path,
                image_path)
                img = io.imread(input_path)
                fts,_ = pywt.dwt2(img, 'haar')
                features.append(fts)
            features = self.flatten (features)
            self.x += features
            self.y += [i for f in features]
            i += 1
        return self.x, self.y
```

2 Modelos Utilizados

2.1 SVM

Support Vector Machines (SVM) son modelos de aprendizaje supervisado con algoritmos de aprendizaje asociados que analizan los datos utilizados para el análisis de clasificación y regresión. En machine learning, el conjunto de datos decide por completo el destino de los algoritmos. SVM es un algoritmo de aprendizaje supervisado que requiere datos limpios y anotados. En nuestro caso, utilizaremos la base de datos proveída en la hoja de proyecto y la librería sklearn para este y los demás modelos.

```
if "svm" in method:
             self.clf = make_pipeline (StandardScaler (),
              LinearSVC(random_state=0,
              tol=1e-5, max_iter=5000))
```

2.2 Decision Tree

Un Decision Tree es una herramienta de apoyo a las decisiones que utiliza un modelo de decisiones en forma de árbol y sus posibles consecuencias, incluidos los resultados de eventos futuros, los costos de los recursos y la utilidad. Es una forma de mostrar un algoritmo que solo contiene declaraciones de control condicionales.

Los árboles de decisiones se utilizan comúnmente en la investigación de operaciones, específicamente en el análisis de decisiones, para ayudar a identificar una estrategia con más probabilidades de alcanzar un objetivo, pero también son una herramienta popular en el aprendizaje automático.

```
        if "tree" in method:
            self.clf=make_pipeline (StandardScaler (),
            DecisionTreeClassifier(random_state=0))
```

2.3 KNN

K-nearest neighbours (KNN) es un método de clasificación no paramétrico utilizado para clasificación y regresión. En ambos casos, la entrada consta de los k ejemplos de entrenamiento más cercanos en el conjunto de datos. El resultado depende de si se utiliza KNN para clasificación o regresión.

Para clasificación, el resultado es una pertenencia a una clase. Un objeto se clasifica mediante un voto de pluralidad de sus vecinos, y el objeto se asigna a la clase más común entre sus k vecinos más cercanos (k es un número entero positivo, típicamente pequeño). Si k = 1, entonces el objeto simplemente se asigna a la clase de ese único vecino más cercano.

```
if "knn" in method:
            self.clf = make_pipeline (StandardScaler (),
            KNeighborsClassifier(n_neighbors=100))
```

3 Experimentación

3.1 K-Fold

Para el primer caso de experimentación probamos los diferentes métodos de clasificación con un k-fold para k = 10. En este caso los parámetros de cada metodo fueron: knn(k=100), svm(iter=5000),dtree() .Donde vació significa que se usaron los parámetros por default. A continuación vemos los resultados de error al tomar el promedio de presiones para cada método. Y los valores exactos de error son 0.72,0.54 y 0.25 respectivamente. Las pruebas tienen la siguiente estructura:

```
method="knn"
clfknn = Classfication (method)
characteristics, emotions = clfknn.generate_datasets ()
k = int (len (characteristics) / 10)
accuracy2 = clfknn.k_fold_cross (characteristics, emotions,
 k, clfknn.clf.fit, clfknn.clf.predict)
print (accuracy2)
```

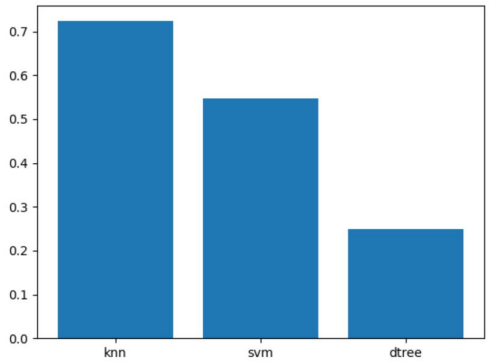

Figure 1: Valores de error para $k = 10$

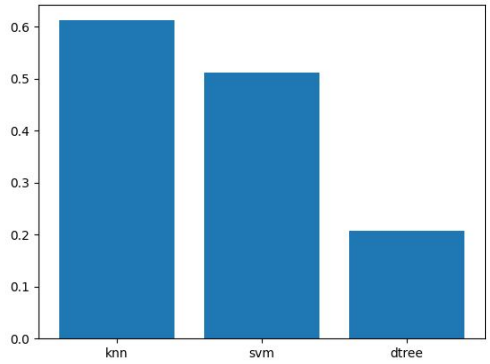

Figure 2: Valores de error para $k = 20$

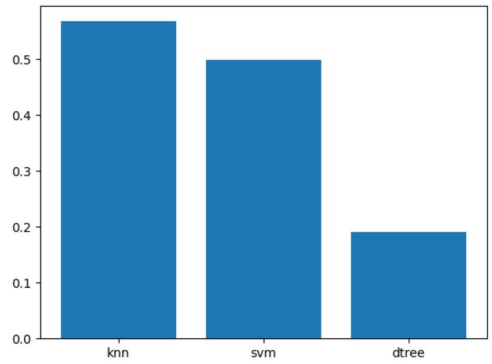

Figure 3: Valores de error para $k = 30$

Como podemos ver, los errores se van disminuyendo conforme vamos aumentando la cantidad de particiones, esto se debe a la distribución de sus imágenes y el hecho de que más particiones mejora la capacidad de los clasificadores de adaptarse al dataset de test.

3.2 Bootstrap

Para el método de bootstrap se toman las misma condiciones mencionadas anteriormente y se defino un número de iteraciones fijo para probar con cada de método. De igual manera se ira variando este parámetro y comparando los resultados.

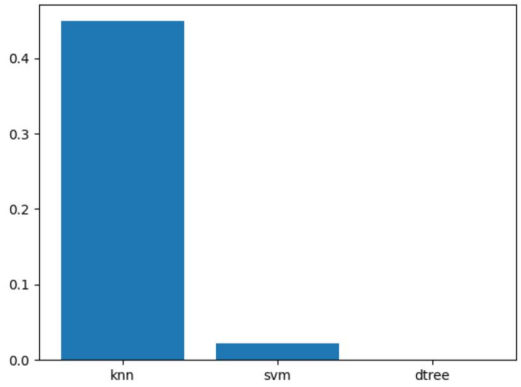

Figure 4: Valores de error para $ite = 5$

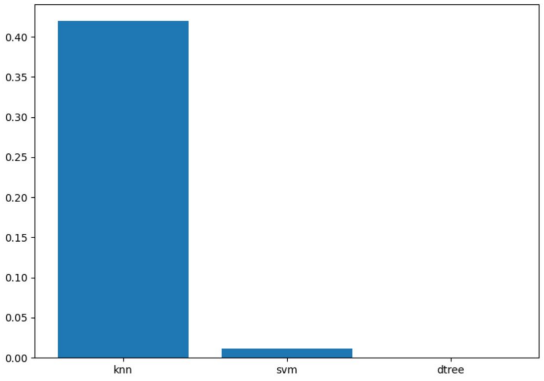

Figure 5: Valores de error para $ite = 10$

Como podemos ver, el Dtree tiene un error de cero para ambos casos. Esto se debe a que el tamaño del test set es muy pequeño y el modelo empieza a sobre ajustarse de manera rápida. Por otro lado el SVM tiene un error cercano a cero que mejora conforme se aumenta la cantidad de iteraciones. Para el knn, existe el parámetro que se tendría que optimizar para que el modelo compita en términos de error con los demás que funcionan bastante bien con los parámetros default.

4 Estimacion de Errores

4.1 K-fold cross validation

El K-fold cross validation es un método estadístico que se utiliza para estimar la habilidad de los modelos de aprendizaje automático.

Se usa comúnmente en el aprendizaje automático aplicado para comparar y seleccionar un modelo para un problema de modelado predictivo dado porque es fácil de entender, fácil de implementar y da como resultado estimaciones de habilidades que generalmente tienen un sesgo más bajo que otros métodos.

4.2 Bootstrap

Bootstrapping es cualquier prueba o métrica que utiliza muestreo aleatorio con reemplazo y se incluye en la clase más amplia de métodos de remuestreo. Bootstrapping asigna medidas de precisión (sesgo, varianza, intervalos de confianza, error de predicción, etc.) a estimaciones de muestra. Esta técnica permite estimar la distribución muestral de casi cualquier estadística utilizando métodos de muestreo aleatorio.

4.3 Varianza de errores

Como sabemos , la varianza del erros puede ser calculada con :

$$\frac{\sum((e_i - e_m)^2)}{n} \tag{1}$$

que corresponde a:

```
def V(accuracy):
    ans=0
    meanerror=1-avg(accuracy)
    for i in accuracy:
        ans=ans+(((1-i)-meanerror))**2
    return ans/len(accuracy)
```

A continuación tenemos los valores calculados para K-Fold con k =10 y Bootstrap sampling con 10 iteraciones

Varianza	SVM	Decision Tree	KNN
K-Fold	0.06	0.04	0.07
Bootstrap	0.001	0.0	0.02

5 Conclusiones

Con respecto a los gráficos, se ha discutido los casos para cada modelo y como se puede notar, en general el método que mejor desempeño tiene es el Decision Tree. Sin embargo, para un caso de modelado real, habría que ser muy cuidadosos con no permitir el sobre ajuste para garantizar una generalización efectiva con los datos no vistos. Esto se debe probablemente a que es el método que funciona mejor con los parámetros puestos por default. Para alcanzar la precisión que tiene este método se tendría que optimizar los demás y modificar sus parámetros.