Predictor De Votaciones

La clasificación es aquel proceso que nos permite asignar objetos a diferentes categorías predefinidas, un modelo de clasificación intenta extraer alguna conclusión de los valores observados. Dadas una o más entradas al modelo de clasificación, intentaremos predecir el valor de uno o más resultados. Los resultados son etiquetas que se pueden aplicar a un conjunto de datos. Por ejemplo, al filtrar correos electrónicos "spam" o "no spam"

En este proyecto se realizaron múltiples modelos de clasificación que nos permiten predecir los votos de las elecciones del año 2018 en Costa Rica (primera ronda, segunda ronda y segunda ronda basandonos en los votos de la primera), basándose para esto en un conjunto de muestras generadas aleatoriamente con los indicadores proporcionados por la PEN (Programa Estado de la Nación) y los votos de la primera y segunda ronda.

Aspectos Técnicos

En esta sección se pretende abarcar las herramientas utilizadas y demás aspectos técnicos del proyecto.

Simulador de votantes

Con este modulo generador de votantes (desarrollado previamente a este proyecto) se generará muestras de votantes en las votaciones de Costa Rica. El generador puede crear muestras tanto para todo el país como para solo cierta provincia.

Sobre los datos de las muestras

Los datos utilizados para generar estas muestra han sido recolectados de las Actas de Sesión de la primera y segunda ronda de elecciones del año 2018 y de los Indicadores Cantonales y Censos del año 2011.

El único dato que se sacó de un archivo externo fue el de rango de edades, para el cual se han utilizado los datos del documento de *Estimaciones y*

Proyecciones de Población por sexo y edad (1950-2100) de la INEC. Más específicamente de la sección *Proyecciones de población del periodo 2000-2050* (cuadros 2.3 y 2.4)

Funcionamiento del simulador

Como fue mencionado el simulador utiliza los datos de Indicadores Cantonales y Censos, para cada votante generado se aletoriza cada una de las propiedades del votante segun estos censos, y para la elección de canton se decide al azar segun los votos de primera ronda. al igual que las propiedaes el voto de la persona generada es establecido de manera aleatoria, esto segun las Actas de Sesión de primera o segunda ronda (segun sea necesario).

Librerías utilizadas

Para la clasificación SVM y manejo de los datos se usó principalmente la librería scikit de python junto con numpy. En la clasificación por Modelos Lineales se utilizó la librería mencionada previamente y la librería de Tensorflow para la creación de los tensores que nos ayudan a la clasificación de los datos por medio de una regresión logística y la librería OneHotEncoder, el cual es un proceso mediante el cual las variables categóricas se convierten en una forma que podría proporcionarse a los algoritmos de modelos lineales para hacer un mejor trabajo en la predicción. La clasificación por Redes Neuronales además de haber utilizado las librerías mencionadas anteriormente con el mismo objetivo, a este se le incorpora la librería de Keras, el cuál nos permite crear las capas de la neurona mediante Dense de keras y un modelo secuencial mediante Sequential de keras.

Reportes

Modelos lineales

Parametros del modelo

Este modelo recibirá como parámetro el tipo de regularización que se quiere aplicar en el modelo, las cuales son L1 y L2.

L1: Este se ingresa por medio de la bandera --l1 y es un nivel de regularización provisto por tensorflow que utiliza la técnica "Lasso Regression" que se aplica a los pesos.

L2: Este se ingresa por medio de la bandera --l2 y es un nivel de regularización provisto por tensorflow que utiliza la técnica "Ridge Regression" que se aplica a los pesos.

Análisis de resultados

Para el análisis de este modelo se utilizarán muestras de tamaño de 100, 1000 y 5000. Para todas se guardará un dos por ciento de las muestras para realizar la prueba final, además se aplicarán la regularización l1 y l2 para cada grupo de muestras con una escala de 0.001, 0.00001 y 0.0000001, con un epoch(iteración sobre todos los datos de entrenamiento) de 800.

Con regularización L1:

1) Regularizacion: l1, scale: 0.001

	100	**1000**	**5000**
Primera ronda	0.25	0.22	0.24
Segunda ronda	0.67	0.59	0.5955
Basado en primera	0.56	0.58	0.5957

2) Regularizacion: l1, scale: 0.00001

	100	**1000**	**5000**
Primera ronda	0.23	0.25	0.25
Segunda ronda	0.71	0.57	0.58
Basado en primera	0.67	0.58	0.59

3) Regularizacion: l1, scale: 0.0000001

	100	**1000**	**5000**
Primera ronda	0.225	0.25	0.23

	100	1000	5000
Segunda ronda	0.712	0.58	0.60
Basado en primera	0.575	0.57	0.58

Con regularización L2::

1) Regularizacion: l2, scale: 0.001

	100	1000	5000
Primera ronda	0.25	0.22	0.24
Segunda ronda	0.67	0.59	0.5955
Basado en primera	0.56	0.58	0.5957

2) Regularizacion: l2, scale: 0.00001

	100	1000	5000
Primera ronda	0.275	0.262	0.237
Segunda ronda	0.512	0.582	0.591
Basado en primera	0.575	0.612	0.595

3) Regularizacion: l2, scale: 0.0000001

	100	1000	5000
Primera ronda	0.262	0.237	0.249
Segunda ronda	0.612	0.616	0.596
Basado en primera	0.562	0.603	0.591

Redes neuronales

Parametros del modelo

El modelo de la Red Neuronal trabaja con 3 parámetros, los cuales son layers, las unit_per_layer y la activation_func. Estos se explican a continuación:

layers: Define la cantidad de capas que se quiere agregar al modelo de la red neuronal para su entrenamiento.

unit_per_layer: Establece las unidades que se le quiere asignar a cada capa agregada anteriormente para asignarle la dimensionalidad del espacio de salida de la capa.

activation_func: Esta es la función de activación que se quiere utilizar en las capaz agregadas, este puede ser 'relu', 'sigmoid' o 'tanh'.

Análisis de resultados

Para el análisis del modelo de Red Neuronal se utilizarán muestras de tamaño 100, 1000 y 5000. Esto guardará un veinte por ciento de las muestras generadas para realizar la prueba final. A cada muestra se le agregará una cantidad de capas de 5, 10 y 20, además de las unidades para cada capa con números aleatorios y una función de activación utilizando 'sigmoid'.

Utilizando función de activación Sigmoid:

1) layers: 5, unit_per_layer: [10,15,20,25,27], activation_func: 'sigmoid'

	100	**1000**	**5000**
Primera ronda	0.15	0.23	0.25
Segunda ronda	0.50	0.56	0.59
Basado en primera	0.60	0.59	0.58

1) layers: 10, unit_per_layer: [10,15,20,25,27,30,32,35,45,50], activation_func: 'sigmoid'

	100	**1000**	**5000**
Primera ronda	0.20	0.23	0.23
Segunda ronda	0.45	0.59	0.59
Basado en primera	0.55	0.58	0.62

1) layers: 20, unit_per_layer: [5,8,10,10,12,15,20,25,27,30,32,35,45,50,52,55,62,68,90,100], activation_func: 'sigmoid'

	100	1000	5000
Primera ronda	0.22	0.25	0.23
Segunda ronda	0.71	0.58	0.60
Basado en primera	0.57	0.57	0.58

Se puede observar que la cantidad de capas puede afectar el "accuracy" de la clasificación por el modelo de redes neuronales.

Árboles de decisión

Parametros del modelo

Este modelo solo recibe un parámetro el cual es el threshold. Este se explicará a continuación.

threshold: este especifíca la ganancia de información mínima requerida para realizar una particón.

Análisis de resultados

Para el análisis del modelo se pretende utilizar muestras de tamaños 100, 1000 y 5000, el cuál se guardará un veinte por ciento para pruebas. Además se utilizará un threshold o umbral de poda de 10, 50 y 100.

Cada prueba muestra el error de entrenamiento (ER) promedio del modelo luego de 30 corridas.

Umbral de poda de 10

1) Cross Validation.

	100	1000	5000
Primera ronda	0.933	0.81	0.75
Segunda ronda	0.766	0.38	0.415
Basado en primera	0.766	0.38	0.415

2) Pruebas

	100	1000	5000
Primera ronda	0.8	0.76	0.71
Segunda ronda	0.65	0.41	0.395
Basado en primera	0.65	0.41	0.395

Umbral de poda de 50.

1) Cross Validation

	100	1000	5000
Primera ronda	0.866	0.79	0.71
Segunda ronda	1.0	0.44	0.36
Basado en primera	1.0	0.44	0.36

2) Pruebas

	100	1000	5000
Primera ronda	0.75	0.73	0.73
Segunda ronda	1.0	0.99	0.39
Basado en primera	1.0	0.99	0.39

Umbral de poda de 100.

1) Cross Validation

	100	1000	5000
Primera ronda	0.9	0.78	0.73
Segunda ronda	1.0	0.43	0.99
Basado en primera	1.0	0.43	0.35

2) Pruebas

	100	1000	5000
Primera ronda	1.0	0.73	0.73
Segunda ronda	1.0	0.98	0.99
Basado en primera	1.0	0.98	0.37

KNN

Parametros del modelo

Este modelo recibe como parámetros el número de vecinos a considerar.

numero: Indica el número de vecinos a considerar para el modelo de "nearest neighbors".

Análisis de resultados

Para el análisis del modelo se pretende utilizar muestras de tamaños 100, 1000. Se guardará un veinte por ciento de las muestras para la realización de la prueba final. Además de utilizará una cantidad de 10, 20, 50 y 100 vecinos más cercanos. Este retorna el error de entrenamiento

10 vecinos más cercanos

1) Cross Validation.

	100	1000
Primera ronda	0.9	0.81
Segunda ronda	0.3	0.47
Basado en primera	0.3	0.34

2) Pruebas

	100	1000
Primera ronda	0.75	0.80
Segunda ronda	0.55	0.45
Basado en primera	0.6	0.44

20 vecinos más cercanos

1) Cross Validation.

	100	**1000**
Primera ronda	0.86	0.80
Segunda ronda	0.46	0.44
Basado en primera	0.46	0.44

2) Pruebas

	100	**1000**
Primera ronda	0.85	0.86
Segunda ronda	0.35	0.52
Basado en primera	0.44	0.50

SVM

Parametros del modelo

El modelo trabaja con tres diferentes parametros, el kernel, C y gamma:

Kernel: Un kernel es una función de similitud. Se proporciona a un algoritmo de aprendizaje automático el cual toma dos entradas y retorna que tan similares son.

C: Intercambia errores de clasificación de ejemplos de entrenamiento contra la simplicidad de la superficie de decisión. Una C baja hace que la superficie de decisión sea suave, mientras que una C alta tiene como objetivo clasificar correctamente todos los ejemplos de entrenamiento. En otras palabras C define cuánto se quiere evitar clasificar erróneamente cada ejemplo.

Gamma: Define cuánta influencia tiene un único ejemplo de entrenamiento. Cuanto más grande es gamma, más cerca deben verse otros ejemplos para ser afectados.

Análisis de resultados

Para el análisis del modelo se pretende utilizar muestras de tamaños 100, 1000, 2500 (solo rbf) y 5000, para todas se guardará un dos por ciento de las muestras para realizar la prueba final. Además, en SVM se probarán los kernel "rbf" y "sigmoid", para los valores de C se probarán valores 1 y 10, para gamma se probarán valores exponenciales de 1 a 0.000000001 y el valor auto (que se calcula segun la cantidad de propiedades).

Cada prueba muestra el error de entrenamiento (ER) promedio del modelo luego de 30 corridas.

Pruebas (rbf):

1) Kernel: rbf, C: 1, Gamma: 1

	100	**1000**	**2500**	**5000**
Primera ronda	0.772	0.77	0.761	0.73
Segunda ronda	0.444	0.41	0.42	0.41
Basado en primera	0.445	0.39	0.41	0.40

2) Kernel: rbf, C: 1, Gamma: 0.000000001

Un gamma bajo, un C bajo y rbf mejoraron ligeramente las predicciones de ronda 1, pero con las demás no se vio una mejora significativa.

	100	**1000**	**2500**	**5000**
Primera ronda	0.792	0.766	0.75	0.7448
Segunda ronda	0.442	0.4	0.4	0.4
Basado en primera	0.442	0.4	0.4	0.4

3) Kernel: rbf, C: 1, Gamma: auto

Al usar un gamma calculado con un algoritmo, un C bajo y rbf se noto que la predicción mejoraba con todas las rondas, lastimosamente los valores de segunda ronda y segunda ronda basado en primera no variaron entre sí por lo que no se ve como los parametros optimos.

	100	**1000**	**2500**	**5000**
Primera ronda	0.762	0.74	0.737	0.7294

	100	1000	2500	5000
Segunda ronda	0.432	0.3975	0.379	0.378
Basado en primera	0.43	0.3975	0.379	0.378

4) Kernel: rbf, C: 10, Gamma: 1

Un gamma alto y un C alto con rbf no es una buena combinación para los datos de segunda ronda, más que todo porque un gamma alto con muchas propiedades hace que las predicciones no sean tan buenas, por otra parte nos permitio ver diferentes resultados en ronda 2 y ronda 2 basado en primera, ya que normalmente estas terminan predicciendo lo mismo.

	100	1000	2500	5000
Primera ronda	0.792	0.743	0.76	0.7638
Segunda ronda	0.385	0.413	0.4192	0.422
Basado en primera	0.385	0.408	0.4113	0.416

5) Kernel: rbf, C: 10, Gamma: 0.000000001

Utilizar un C alto y un gamma lo suficientemente bajo con rbf mejora en gran cantidad con muchos valores de predicción como lo fue ronda 1, en cambio con ronda 2 o ronda 2 basado en primera no se nota una muy buena mejora en comparación a usar un algoritmo para calcular gamma.

	100	1000	2500	5000
Primera ronda	0.81	0.76	0.7442	0.726
Segunda ronda	0.415	0.41	0.3964	0.384
Basado en primera	0.415	0.41	0.3969	0.384

6) Kernel: rbf, C: 10, Gamma: auto

La utilización de un C alto y un algoritmo que saque Gamma segun la cantidad de propiedades demostro que el predictor mejorara entre más muestras reciba, como se puede notar la diferencia entre hacer la prediccion con 100 muestras y hacerlo con 5000, el error disminuye en más de un 5%. Tambien en estos datos influye el uso del kernel rbf pues con sigmoid no se nota esta mejora, como se vera más adelante.

	100	1000	2500	5000
Primera ronda	0.78	0.76	0.7481	0.7388
Segunda ronda	0.447	0.41	0.3953	0.387
Basado en primera	0.457	0.407	0.3982	0.389

Con los anteriores parametros se noto que el clasificador mejoraba, pues el error de entrenamiento iba dismiuyendo conforme de usaban más muestras.

Pruebas (sigmoid):

1) Kernel: sigmoid, C: 1, Gamma: 1

	100	1000	5000
Primera ronda	0.735	0.7455	0.75
Segunda ronda	0.417	0.41	0.4
Basado en primera	0.417	0.41	0.4

2) Kernel: sigmoid, C: 1, Gamma: 0.000000001

	100	1000	5000
Primera ronda	0.783	0.806	0.803
Segunda ronda	0.433	0.489	0.485
Basado en primera	0.433	0.489	0.485

3) Kernel: sigmoid, C: 1, Gamma: auto

	100	1000	5000
Primera ronda	0.785	0.75	0.757
Segunda ronda	0.397	0.387	0.4
Basado en primera	0.397	0.387	0.4

4) Kernel: sigmoid, C: 10, Gamma: 1

	100	1000	5000
Primera ronda	0.775	0.757	0.755
Segunda ronda	0.398	0.409	0.3937
Basado en primera	0.398	0.409	0.3937

5) Kernel: sigmoid, C: 10, Gamma: 0.000000001

	100	1000	5000
Primera ronda	0.82	0.8	0.81
Segunda ronda	0.51	0.47	0.48
Basado en primera	0.51	0.47	0.48

6) Kernel: sigmoid, C: 10, Gamma: auto

	100	1000	5000
Primera ronda	0.772	0.75	0.757
Segunda ronda	0.428	0.408	0.4
Basado en primera	0.428	0.408	0.4

Manual de usuario

Instalación de Librerías de Python

Para el funcionamiento de este programa se necesita instalar los siguientes módulos de python los cuales se instalan desde la terminal del sistema operativo:

1. Tensorflow
    - Si se desea instalar en linux, seguir los siguientes pasos e ingresar los comandos en la terminal.
        a. Instalar pip o el ambiente virtual.

```
$ sudo apt-get install python-pip python-dev pytl
$ sudo apt-get install python3-pip python3-dev py
```

b. Crear el ambiente virtual.

```
$ virtualenv --system-site-packages targetDirecto
$ virtualenv --system-site-packages -p python3 ta
```

c. Activar el ambiente virtual.

```
$ source ~/tensorflow/bin/activate # bash, sh, ks
$ source ~/tensorflow/bin/activate.csh  # csh or
$ . ~/tensorflow/bin/activate.fish  # fish

Al hacer esto, la raíz de la terminal debería de

(tensorflow)$
```

d. Nos aseguramos de que pip esté instalado.

```
(tensorflow)$ easy_install -U pip
```

e. Elegir alguno de los siguientes comandos para instalar Tensorflow.

```
(tensorflow)$ pip install --upgrade tensorflow
(tensorflow)$ pip3 install --upgrade tensorflow
(tensorflow)$ pip install --upgrade tensorflow-gp
(tensorflow)$ pip3 install --upgrade tensorflow-g
```

- Si se desea instalar en Windows se puede utilizar el pip nativo de python para instalarlo. Para esto escribir el siguiente comando en la terminal:

```
pip3 install --upgrade tensorflow
```

2. Keras

Este se puede instalar de dos maneras:
- Si es desde PyPi, escribir en la terminal:

```
sudo pip install keras
```

- Si se está utilizando un ambiente virtual, escribir en la terminal:

```
pip install keras
```

3. Numpy

   Este es parte de la instalación de python, pero en caso de no tenerlo, se puede escribir en la terminal el siguiente comando:

```
pip install numpy
```

4. sklearn

   Este módulo se puede instalar mediante pip nativo de python ingresando el siguiente comando en la terminal:

```
pip install -U scikit-learn
```

Instalación del simulador de votos

Además de instalar los módulos anteriores, se debe de instalar el simulador de votantes de la siguiente manera:

1. Abrir terminal del sistema operativo.
2. Dirigirse a la carpeta en donde se encuentra el archivo setup.py. Por Ejemplo:

```
cd C:\Users\user\Desktop\PredictorDeVotaciones
```

3. Ingresar el siguiente comando:

```
python install setup.py
```

Al haber hecho los pasos anteriores se podrá utilizar el simulador de votantes para generar muestras de los votantes de Costa Rica.

Utilización del Predictor de Votos

Para la utilización del Predictor de Votos se deben seguir los siguientes pasos:

1. Abrir terminal del sistema operativo.
2. Dirigirse a la carpeta llamada p1, la cual se encuentra en la siguiente ruta:

```
.\PredictorDeVotaciones\tec\ic\ia\p1
```

3. Ingresar el siguiente comando:

```
main.py
```

   Este desplegará en pantalla las instrucciones para la utilización del programa. Este se ve de la siguiente manera:

```
***INSTRUCCIONES***

main.py --poblacion<poblacion> --porcentaje

BANDERAS:

*    --regresion-logistica [--l1 o --l2]
*    --red-neuronal --numero-capas <numero> --un
*    --knn --k <numero de vecinos>
*    --arbol --umbral-poda <numero>
*    --svm --kernel <linear, poly, rbf, sigmoid>
```

4. Ingresar cualquiera de los comandos mostrados según la clasificación que se desee realizar.