Resumen

El proyecto consiste en hacer clasificación binaria para determinar si una persona sobrevive(y=1) o no (y=0) del hundimiento del Titanic .
El enunciado parece ser de un proyecto largo , pero es por que buscamos hacerlo detallado y poner ejemplos o referencias de cosas que pueden facilitar el trabajo.

En este proyecto se busca crear un modelo con u**n nivel de exactitud de al menos el 80%**(aunque es posible crear modelos mejores que esto).

El proyecto está dividido en 2 partes de manera similar a como se divide o se trabaja un proyecto de ML real, es decir.

1 **Entrenamiento,selección y validación.**

- En esta parte se busca realizar el entrenamiento de los modelos y todo lo que esto conlleva:
    - selección y transformación de features (feature eng.) . Recordar usar características generales , no específicas como identificadores o nombres.
    - hacer las validaciones correspondientes(usando métricas de evaluación) y selección de hyper-parámetros y modelos.

En este proyecto haremos “ensemble” learning con 4 tipos de modelos : árbol de decisión, svm , naive bayes y regresión lineal (con regularización).

- Los primeros 2 con scikit learn y su conveniente y famosa función fit(x,y) para ahorrar tiempo y simplificar.
- Naive bayes con numpy o pandas (como recordatorio naive bayes consiste en aplicar el teorema de bayes basado en tablas de frecuencia o probabilidad que no son más que un tensor)
- Regresión logística con tensorflow: parecido pero más sencillo que el de la tarea previa ya que no necesita softmax,usamos sigmoid (ni matriz de pesos, solo vector de pesos) por ser clasificación binaria. Lo nuevo es: agregar regularización para evitar sobre-ajuste

Por razones de tiempo y de estar combinando distintas herramienta no es obligatorio usar tensorboard(opcional para quien desee hacerlo)

2 **Inferencia, predicción y despliegue de modelos**
En ML una vez entrenado el modelo (o modelos) estos son usados para predicción (etapa de inferencia) sobre nuevas observaciones,comúnmente integrandolos a una aplicación de software (por ejemplo mobile o web) a través de un proceso conocido como “**despliegue o deployment de ML”.**
Por ejemplo: youtube y sus recomendaciones, waze y su predicción de tiempo en el tráfico, o el proyecto de los alumnos de la UFM : https://ml-ufm-20018.firebaseapp.com/

En este proyecto simularemos de manera sencilla este proceso a través de usar un segundo notebook de jupyter en el cual cargaremos los modelos elegidos y generaremos predicciones sobre nuevos x con estos.

Descripción detallada

Parte 1: Entrenamiento y validación y selección(Nivel de exactitud mínimo deseado: 80%**)**

Nota:
En los casos en donde se deba calcular métricas de evaluación,con objetivos de simplificar el problema se puede usar cualquier herramienta , por ejemplo sklearn en el subpaquete metrics:

- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
- https://scikit-learn.org/stable/modules/model_evaluation.html

Train-val-test split
El primer paso es separar los datos en entrenamiento, validación y pruebas:

- separar datos en entrenamiento y pruebas. Por ejemplo usando: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- tomar una porción de datos de entrenamiento del paso anterior para validación. Puede ser aplicando nuevamente https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Esto genera en resumen:

1. Conjunto de entrenamiento
2. Conjunto de validación
3. Conjunto de pruebas.

Nota: Guardar y NO USAR LOS DATOS DE PRUEBAS HASTA EL FINAL(solo entrenamiento y validación)

Ensemble learning:
Se aplicará ensemble learning para crear una "votación mayoritaria" con distintos tipos de modelos, con el objetivo de simplificar el problema 2 de estos modelos se harán usando scikit-learn y su función .fit(x,y)

Para simplificar el problema , **no hacer muestreo bootstrap**, en todos los modelos usar los mismos datos de entrenamiento(en problemas reales sí se recomienda hacer bootstrapping).

No se hace bootstrapping pero se debe agregar un comentario en markdown de cómo se podría haber hecho si en caso se hubiera requerido en este proyecto.

El ensamble estará compuesto de : :

- Árbol de decisión con sklearn
- SVM con sklearn
- Naive bayes con numpy y/o pandas

- Reg. logística binaria(sigmoid) en Tensorflow con regularización (probar L1, L2 y distintos valores del factor de regularización y elegir el mejor) y mini-batch gradient descent usando tamaño de mini-batch como hyper-parametro.

Se recomienda crear una función de Python para el entrenamiento de cada tipo de modelo que reciba los parámetros adecuados para cada uno y devuelva el modelo entrenado y cualquier otra información necesaria y relevante
Por ejemplo para SVM:

```
def train_SVM(X,Y, C,lr ):
  …
  …
  return svm_model
```

Cada vez que se entrena un modelo (si se usan funciones como la anterior, cada vez que se ejecuta una de estas funciones) corresponde a un experimento, y para cada experimento(o llamada a una de las funciones):

- Debemos crear la cadena de configuración que describa cómo se realizó cada experimento, variables usadas, valores de hyper-parámetros y tipo de modelo usado (similar al config string que hemos usado en tensorboard) por ejemplo:
  - Se entrena un regresión .logística con lr=0.01 ,factor de regularización = 0.1 usando las variables variables var1,var2,var3, la cadena podría ser:
    regLog_lr=0.01_reg=0.1_var1_var2_var3
    Esta cadena nos servirá después para identificar cada experimento, y anotar en una bitácora(excel o csv) las métricas de evaluación de cada uno.
  - Cada uno de los 4 tipos de modelos pueden requerir más de 1 experimento, con distintos hyper-parámetros o variables usadas, diagnosticamos overfitting u underfitting y realizamos acciones para atacarlos.
- Por cada experimento debemos guardar en un excel o csv la cadena de configuración y las métricas de evaluación : accuracy,error, precisión,recall,f1-score , estas serán evaluados en el dataset de entrenamiento y el de validación. Es posible usar sklearn(o cualquier otro método que facilite el cálculo) para las métricas de evaluación y Pandas para guardar el csv o excel.
  Este excel o csv no debe ser sobreescrito en cada experimento si no que deben agregarse nuevas filas, esta será la bitácora de experimentos.
- Por cada experimento debemos guardar el modelo correspondiente identificandolo con su cadena de configuración, por ejemplo :
  - con sklearn para guardar un modelo "model" del experimento svm_lr=0.01_reg=0.1_var1_var2_var3 se puede usar:
    from sklearn.externals import joblib

    joblib.dump(model, 'svm_lr=0.01_reg=0.1_var1_var2_var3.pkl')
  - Con tensorflow podemos ejecutar en la session los parámetros entrenables(al terminar el entrenamiento) y asignarlos a un tensor de Numpy llamado W,
    luego podemos guardar el tensor de parámetros del modelo con dump()
    W = session.run(parametros,feed_dict)

    W.dump("regLog_lr=0.01_reg=0.1_var1_var2_var3.npy")
  - Con numpy (para el clasificador de bayes) podemos también usar dump (o si se usa pandas usar to_csv por ejemplo)

Luego de haber ejecutado ,evaluado, y guardado varios experimentos,elegimos para cada tipo de modelo el mejor basado en las métricas de evaluación sobre los datos de validación. (recordar la exactitud deseada)

Investigar la técnica: k-fold cross validation agregando en markdown una descripción de esta y como se pudo haber aplicado en este proyecto (No debemos aplicarla al proyecto , solo describir la técnica y cómo se podría aplicar) . Por ejemplo: https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f

Prueba/Evaluación final:

Dada el conjunto de observaciones X de el conjunto de pruebas y los 4 modelos elegidos:

- Predecir sobre estas usando el mejor modelo de cada tipo elegido en el punto anterior.
- Combinar los resultados de las predicciones en una predicción final(moda de resultados individuales)
- Generar una tabla de predicciones como el ejemplo x y crear un dataframe de Pandas con los resultados.
- Calcular métricas de evaluación comparando los Y reales del conjunto de pruebas, contra él Y que se obtuvo de combinar las predicciones individuales del modelo.
- Similar al paso anterior, generar una tabla de métricas de evaluación y crear un dataframe de pandas para mostrar el resultado final.

  Si en la evaluación final, no se obtiene la exactitud mínima deseada, volver a la fase de experimentación .

Conclusiones
Agregar sección de conclusiones y recomendaciones (incluyendo opiniones,experiencias ,dificultades y lecciones aprendidas).

Parte 2(Deployment , inferencia y predicción) :

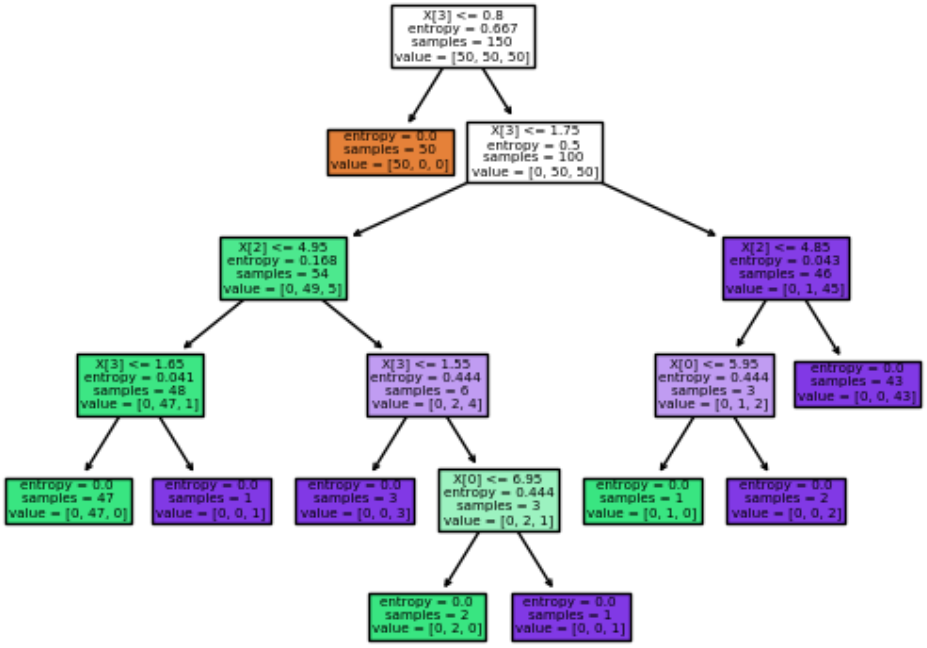
En un notebook distinto e independiente del usado "simular" como el modelo (resultado de combinar los mejores 4) puede ser usado para realizar predicciones en nuevos datos.

Para esto:

- Cargar los modelos elegidos :
  - Si es sklearn usar :
    svm_model = joblib.load('svm_lr=0.01_reg=0.1_var1_var2_var3.pkl')
  - Si es tf y se exportó el tensor de parámetros con numpy(o bien si se usó numpy para bayes):
    reg_model = np.load("regLog_lr=0.01_reg=0.1_var1_var2_var3.npy")
  - Si se usó pandas para guardar en csv, usar pd.read_csv

- Crear una función que prediga para cierta observación (una sola) x la predicción y_hat combinada y además:
  - Muestre en el notebook el árbol de decisión y como este llega a una conclusión usando sklearn(ya lo hace) https://scikit-

learn.org/stable/modules/tree.html



- Muestre la predicción probabilística de los modelos de bayes y regresión logística.

- Probar la función anterior para 10 observaciones x distintas(pueden ser del dataset original o inventadas.), de una en una

Fecha de entrega: hasta que la escuela permita subir notas, tentativamente 8 julio