1. Pre-Processing:

Considering that Name and Ticket Number are having too much unique values, it might be better to remove them from the training dataset. Similarly, Cabin Number has too much null numbers, so we also remove that here. (If those are necessary, could be added back in later testing.)

In [138…
```
import tensorflow as tf
import pandas as pd
import numpy as np
#Use Scikit-learn to do the preprocessing since it contains text label
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

In [139…
```
train_file_path = "train.csv"
original_df = pd.read_csv(train_file_path)
original_df
```

Out[139…

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

First, Drop the useless columns.

In [140…
```
#Replace PassengerID with ID as feature name
df = original_df.copy()
df.rename(columns={"PassengerId": "ID"}, inplace=True)
df = df.drop(columns=["Name", "Ticket", "Cabin"])
df
```

Out[140…

	ID	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...	...
886	887	0	2	male	27.0	0	0	13.0000	S
887	888	1	1	female	19.0	0	0	30.0000	S
888	889	0	3	female	NaN	1	2	23.4500	S
889	890	1	1	male	26.0	0	0	30.0000	C
890	891	0	3	male	32.0	0	0	7.7500	Q

891 rows × 9 columns

Check if there has any NA values within the dataframe

In [141…
```
print(df.isna().sum())
```

```
ID            0
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

Since the "Embarked" feature has only 2 NA values, I think it will be better just remove these 2 rows for now.(Could also fill them with some of values instead. Remained to be tested later.).

In [142…
```
df = df.dropna(subset=["Embarked"])
df
```

Out[142…

	ID	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S
…	…	…	…	…	…	…	…	…	…
886	887	0	2	male	27.0	0	0	13.0000	S
887	888	1	1	female	19.0	0	0	30.0000	S
888	889	0	3	female	NaN	1	2	23.4500	S
889	890	1	1	male	26.0	0	0	30.0000	C
890	891	0	3	male	32.0	0	0	7.7500	Q

889 rows × 9 columns

In terms of the age, since it misses 177 values, it is not good just remove them. Other ways of filling in an appropriate value could be tested. But here, we only test the feasibility of the Influence function with titanic. So I assign 0 to them for missing values, and to ensure that it won't affect the other ages. I also add one MissAge feature to say if the age is missing or not.

In [143…
```
#Why copy here is to escape from some warnings.
df = df.copy()
df['MissAge'] = df['Age'].isna().astype(int)
df.fillna({'Age':0}, inplace=True)
df
```

Out[143…

	ID	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	MissAge
0	1	0	3	male	22.0	1	0	7.2500	S	0
1	2	1	1	female	38.0	1	0	71.2833	C	0
2	3	1	3	female	26.0	0	0	7.9250	S	0
3	4	1	1	female	35.0	1	0	53.1000	S	0
4	5	0	3	male	35.0	0	0	8.0500	S	0
…	…	…	…	…	…	…	…	…	…	…
886	887	0	2	male	27.0	0	0	13.0000	S	0
887	888	1	1	female	19.0	0	0	30.0000	S	0
888	889	0	3	female	0.0	1	2	23.4500	S	1
889	890	1	1	male	26.0	0	0	30.0000	C	0
890	891	0	3	male	32.0	0	0	7.7500	Q	0

889 rows × 10 columns

In [144…
```
print(df.isna().sum())
```

```
ID          0
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
MissAge     0
dtype: int64
```

Test again and now we have no NA values.

Now we can do the rest of processing. First, transform text features to numerical features.

In [145…
```
#Change sex and Embarked into numerical
sex_trans = LabelEncoder()
df["Sex"] = sex_trans.fit_transform(df["Sex"])

Emb_trans = LabelEncoder()
df["Embarked"] = sex_trans.fit_transform(df["Embarked"])

df
```

Out[145…

	ID	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	MissAge
0	1	0	3	1	22.0	1	0	7.2500	2	0
1	2	1	1	0	38.0	1	0	71.2833	0	0
2	3	1	3	0	26.0	0	0	7.9250	2	0
3	4	1	1	0	35.0	1	0	53.1000	2	0
4	5	0	3	1	35.0	0	0	8.0500	2	0
…	…	…	…	…	…	…	…	…	…	…
886	887	0	2	1	27.0	0	0	13.0000	2	0
887	888	1	1	0	19.0	0	0	30.0000	2	0
888	889	0	3	0	0.0	1	2	23.4500	2	1
889	890	1	1	1	26.0	0	0	30.0000	0	0
890	891	0	3	1	32.0	0	0	7.7500	1	0

889 rows × 10 columns

Then we normalize the data.

In [146…
```
Normalize = StandardScaler()
Normalize_cols = ["Age","Fare"]
df[Normalize_cols] = Normalize.fit_transform(df[Normalize_cols])
df
```

Out[146…

	ID	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	MissAge
0	1	0	3	1	-0.099150	1	0	-0.500240	2	0
1	2	1	1	0	0.812389	1	0	0.788947	0	0
2	3	1	3	0	0.128735	0	0	-0.486650	2	0
3	4	1	1	0	0.641476	1	0	0.422861	2	0
4	5	0	3	1	0.641476	0	0	-0.484133	2	0
…	…	…	…	…	…	…	…	…	…	…
886	887	0	2	1	0.185706	0	0	-0.384475	2	0
887	888	1	1	0	-0.270063	0	0	-0.042213	2	0
888	889	0	3	0	-1.352516	1	2	-0.174084	2	1
889	890	1	1	1	0.128735	0	0	-0.042213	0	0
890	891	0	3	1	0.470562	0	0	-0.490173	1	0

889 rows × 10 columns

Now we can get prepared to do the training

In [147…
```
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

In [148…
```
X = df.drop(columns=["Survived"])
y = df["Survived"]
IDs = X["ID"].values.reshape(-1, 1).astype(np.float32)
IDs = IDs  / 1e7

X = X.drop(columns=["ID"]).values.astype(np.float32)
X = np.hstack((X, IDs))
y = to_categorical(y.values, num_classes=2)

X.shape
```

Out[148… (889, 9)

In [149…
```
y.shape
```

Out[149… (889, 2)

In [151…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(711, 9)
(178, 9)
(711, 2)
(178, 2)
```

In [153…
```
train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```

In [154…
```
print(len(train_ds))
print(len(test_ds))
```

```
711
178
```

Now we can use a really easy Neural Network for testing.

In [155…
```
from keras import Sequential
from keras import layers
```

In [175…
```
model = Sequential([
    layers.Dense(32, activation="relu", input_shape=(9,)),
    layers.Dense(16, activation="relu"),
    layers.Dense(2, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
model.fit(train_ds.batch(32), epochs=10, validation_data=test_ds.batch(32), verbose=2)
model.evaluate(test_ds.batch(32), verbose=2)
```

```
Epoch 1/10
23/23 - 1s - loss: 0.6454 - accuracy: 0.6203 - val_loss: 0.6399 - val_accuracy: 0.6236 - 550ms/epoch - 24ms/step
Epoch 2/10
23/23 - 0s - loss: 0.6185 - accuracy: 0.6456 - val_loss: 0.6177 - val_accuracy: 0.6517 - 101ms/epoch - 4ms/step
Epoch 3/10
23/23 - 0s - loss: 0.6004 - accuracy: 0.6709 - val_loss: 0.6013 - val_accuracy: 0.6685 - 102ms/epoch - 4ms/step
Epoch 4/10
23/23 - 0s - loss: 0.5857 - accuracy: 0.6962 - val_loss: 0.5871 - val_accuracy: 0.6854 - 95ms/epoch - 4ms/step
Epoch 5/10
23/23 - 0s - loss: 0.5720 - accuracy: 0.7032 - val_loss: 0.5734 - val_accuracy: 0.7135 - 95ms/epoch - 4ms/step
Epoch 6/10
23/23 - 0s - loss: 0.5591 - accuracy: 0.7145 - val_loss: 0.5608 - val_accuracy: 0.7303 - 101ms/epoch - 4ms/step
Epoch 7/10
23/23 - 0s - loss: 0.5469 - accuracy: 0.7286 - val_loss: 0.5488 - val_accuracy: 0.7360 - 90ms/epoch - 4ms/step
Epoch 8/10
23/23 - 0s - loss: 0.5350 - accuracy: 0.7370 - val_loss: 0.5361 - val_accuracy: 0.7472 - 88ms/epoch - 4ms/step
Epoch 9/10
23/23 - 0s - loss: 0.5225 - accuracy: 0.7440 - val_loss: 0.5236 - val_accuracy: 0.7697 - 90ms/epoch - 4ms/step
Epoch 10/10
23/23 - 0s - loss: 0.5099 - accuracy: 0.7581 - val_loss: 0.5117 - val_accuracy: 0.7865 - 97ms/epoch - 4ms/step
6/6 - 0s - loss: 0.5117 - accuracy: 0.7865 - 24ms/epoch - 4ms/step
```

Out[175… [0.5116581320762634, 0.7865168452262878]

Now we have the result from the model, we can test it with influenciae now. Tutorial version below:

In [176…
```
from deel.influenciae.common import InfluenceModel, ExactIHVP
from deel.influenciae.influence import FirstOrderInfluenceCalculator
from deel.influenciae.utils import ORDER
from deel.influenciae.trac_in import TracIn
from keras.losses import BinaryCrossentropy
import warnings
warnings.filterwarnings('ignore')
```

In [177…

```
unreduced_loss = BinaryCrossentropy(from_logits=True, reduction=tf.keras.losses.Reduction.NONE)
influence_model = InfluenceModel(model, start_layer=-1, loss_function=unreduced_loss)

#Tutorial one
ihvp_calculator = ExactIHVP(influence_model, train_dataset=train_ds.shuffle(100).batch(4))
influence_calculator = FirstOrderInfluenceCalculator(influence_model, train_ds.batch(8), ihvp_calculator)

samples_to_explain = test_ds.take(5).batch(1)
explanation_ds = influence_calculator.top_k(samples_to_explain, train_ds.batch(8), k=3, order=ORDER.DESCENDING)

# Print influential samples
for (sample, label), top_k_values, top_k_samples in explanation_ds.as_numpy_iterator():
    #Remember to use round instead of int :<
    sample_id = round(sample[0][-1] * 1e7)
    sample_original = original_df[original_df['PassengerId'] == sample_id]
    print(f"\nTest Sample ID: {sample_id}")
    print("Original sample from DataFrame:")
    print(sample_original[["Survived"]])
    influential_ids = [round(s[-1] * 1e7) for s in top_k_samples[0]]
    for i, (inf_id, score) in enumerate(zip(influential_ids, top_k_values[0])):
        inf_sample_original = original_df[original_df['PassengerId'] == inf_id]
        print(f"Influential Sample {i+1} -> ID: {inf_id}, Influence Score: {score}")
        print(inf_sample_original[["Survived"]])
```

```
Test Sample ID: 282
Original sample from DataFrame:
     Survived
281         0
Influential Sample 1 -> ID: 710, Influence Score: 1.607370138168335
     Survived
709         1
Influential Sample 2 -> ID: 66, Influence Score: 1.606073021888733
    Survived
65         1
Influential Sample 3 -> ID: 302, Influence Score: 1.541520118713379
     Survived
301         1

Test Sample ID: 436
Original sample from DataFrame:
     Survived
435         1
Influential Sample 1 -> ID: 528, Influence Score: 11.818316459655762
     Survived
527         0
Influential Sample 2 -> ID: 803, Influence Score: 10.953840255737305
     Survived
802         1
Influential Sample 3 -> ID: 551, Influence Score: 10.747941017150879
     Survived
550         1
```

```
Test Sample ID: 40
Original sample from DataFrame:
    Survived
39         1
Influential Sample 1 -> ID: 126, Influence Score: 19.923812866210938
     Survived
125         1
Influential Sample 2 -> ID: 831, Influence Score: 16.916419982910156
     Survived
830         1
Influential Sample 3 -> ID: 470, Influence Score: 15.174063682556152
     Survived
469         1

Test Sample ID: 419
Original sample from DataFrame:
     Survived
418         0
Influential Sample 1 -> ID: 693, Influence Score: 2.571444511413574
     Survived
692         1
Influential Sample 2 -> ID: 644, Influence Score: 2.571444511413574
     Survived
643         1
Influential Sample 3 -> ID: 660, Influence Score: 2.446279525756836
     Survived
659         0

Test Sample ID: 586
Original sample from DataFrame:
     Survived
```

```
585         1
Influential Sample 1 -> ID: 551, Influence Score: 14.002045631408691
     Survived
550         1
Influential Sample 2 -> ID: 446, Influence Score: 9.499378204345703
     Survived
445         1
Influential Sample 3 -> ID: 540, Influence Score: 8.959556579589844
     Survived
539         1
```

Also, TraceIn one here

In [178…

```python
#TraceIn one
model = Sequential([
    layers.Dense(32, activation="relu", input_shape=(9,)),
    layers.Dense(16, activation="relu"),
    layers.Dense(2, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

epochs = 10
unreduced_loss_fn = BinaryCrossentropy(from_logits=True, reduction=tf.keras.losses.Reduction.NONE)
model_list = []
model_list.append(InfluenceModel(model, start_layer=-1, loss_function=unreduced_loss_fn))
for i in range(epochs):
```

```
  model.fit(train_ds.batch(32), epochs=1, validation_data=test_ds.batch(32), verbose=2)
  updated_model = tf.keras.models.clone_model(model)
  updated_model.set_weights(model.get_weights())
  model_list.append(InfluenceModel(model, start_layer=-1, loss_function=unreduced_loss_fn))
model.evaluate(test_ds.batch(32), verbose=2)
```

```
23/23 - 0s - loss: 0.6977 - accuracy: 0.5260 - val_loss: 0.6572 - val_accuracy: 0.7022 - 317ms/epoch - 14ms/step
23/23 - 0s - loss: 0.6377 - accuracy: 0.6695 - val_loss: 0.6254 - val_accuracy: 0.6629 - 103ms/epoch - 4ms/step
23/23 - 0s - loss: 0.6106 - accuracy: 0.6639 - val_loss: 0.6015 - val_accuracy: 0.6910 - 106ms/epoch - 5ms/step
23/23 - 0s - loss: 0.5902 - accuracy: 0.6920 - val_loss: 0.5797 - val_accuracy: 0.7303 - 93ms/epoch - 4ms/step
23/23 - 0s - loss: 0.5711 - accuracy: 0.7018 - val_loss: 0.5607 - val_accuracy: 0.7416 - 102ms/epoch - 4ms/step
23/23 - 0s - loss: 0.5536 - accuracy: 0.7089 - val_loss: 0.5449 - val_accuracy: 0.7640 - 92ms/epoch - 4ms/step
23/23 - 0s - loss: 0.5370 - accuracy: 0.7426 - val_loss: 0.5306 - val_accuracy: 0.7809 - 93ms/epoch - 4ms/step
23/23 - 0s - loss: 0.5226 - accuracy: 0.7482 - val_loss: 0.5182 - val_accuracy: 0.7921 - 91ms/epoch - 4ms/step
23/23 - 0s - loss: 0.5103 - accuracy: 0.7623 - val_loss: 0.5084 - val_accuracy: 0.7978 - 91ms/epoch - 4ms/step
23/23 - 0s - loss: 0.4994 - accuracy: 0.7651 - val_loss: 0.4998 - val_accuracy: 0.8034 - 94ms/epoch - 4ms/step
6/6 - 0s - loss: 0.4998 - accuracy: 0.8034 - 21ms/epoch - 4ms/step
```

Out[178… [0.49978184700012207, 0.8033707737922668]

In [179…

```
influence_calculator = TracIn(model_list, 0.01)

samples_to_explain = test_ds.take(5).batch(1)
explanation_ds = influence_calculator.top_k(samples_to_explain, train_ds.batch(8), k=3, order=ORDER.DESCENDING)

# Print influential samples
for (sample, label), top_k_values, top_k_samples in explanation_ds.as_numpy_iterator():
    #Remember to use round instead of int :<
    sample_id = round(sample[0][-1] * 1e7)
    sample_original = original_df[original_df['PassengerId'] == sample_id]
    print(f"\nTest Sample ID: {sample_id}")
    print("Original sample from DataFrame:")
    print(sample_original[["Survived"]])
    influential_ids = [round(s[-1] * 1e7) for s in top_k_samples[0]]
    for i, (inf_id, score) in enumerate(zip(influential_ids, top_k_values[0])):
        inf_sample_original = original_df[original_df['PassengerId'] == inf_id]
        print(f"Influential Sample {i+1} -> ID: {inf_id}, Influence Score: {score}")
        print(inf_sample_original[["Survived"]])
```

```
Test Sample ID: 282
Original sample from DataFrame:
     Survived
281         0
Influential Sample 1 -> ID: 679, Influence Score: 0.08331404626369476
     Survived
678         0
Influential Sample 2 -> ID: 439, Influence Score: 0.07800964266061783
     Survived
438         0
Influential Sample 3 -> ID: 639, Influence Score: 0.07661770284175873
     Survived
638         0

Test Sample ID: 436
Original sample from DataFrame:
     Survived
435         1
Influential Sample 1 -> ID: 234, Influence Score: 0.04841196909546852
     Survived
233         1
Influential Sample 2 -> ID: 775, Influence Score: 0.04238380491733551
     Survived
774         1
Influential Sample 3 -> ID: 262, Influence Score: 0.03728603571653366
     Survived
261         1

Test Sample ID: 40
Original sample from DataFrame:
    Survived
39         1
Influential Sample 1 -> ID: 262, Influence Score: 0.39595136046409607
     Survived
261         1
Influential Sample 2 -> ID: 234, Influence Score: 0.29284849762916565
     Survived
233         1
Influential Sample 3 -> ID: 339, Influence Score: 0.25750496983528137
     Survived
338         1

Test Sample ID: 419
Original sample from DataFrame:
     Survived
418         0
Influential Sample 1 -> ID: 679, Influence Score: 0.09109848737716675
     Survived
678         0
Influential Sample 2 -> ID: 439, Influence Score: 0.08911694586277008
     Survived
438         0
Influential Sample 3 -> ID: 28, Influence Score: 0.08490361273288727
    Survived
27         0

Test Sample ID: 586
Original sample from DataFrame:
     Survived
585         1
Influential Sample 1 -> ID: 234, Influence Score: 0.047035858035087585
     Survived
233         1
Influential Sample 2 -> ID: 775, Influence Score: 0.042571891099214554
     Survived
774         1
Influential Sample 3 -> ID: 438, Influence Score: 0.03695863112807274
     Survived
437         1
```