Titanic - Machine Learning from Disaster

CHARLEMAGNE Clément

January 28, 2024

CarolusCharlemagne

Student Data/IA/C++ at Dassault Aviation
Pélissanne, Provence-Alpes-Côte d'Azur, France
Joined 2 months ago · last seen in the past day

Competitions
Novice

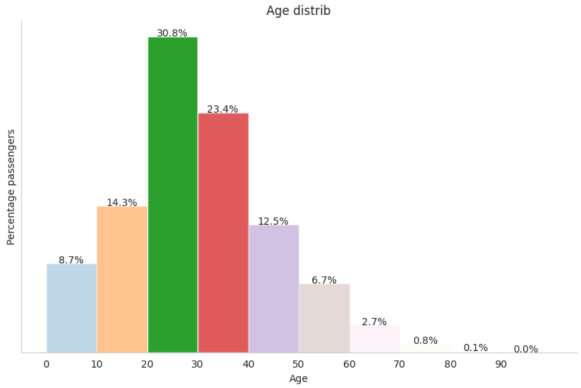
Home | Competitions (1) | Datasets | Code (1) | Discussion | Followers | Notifications | Account | Edit Public Profile

Active | Completed | Hosted | Community | Bookmarks | Default

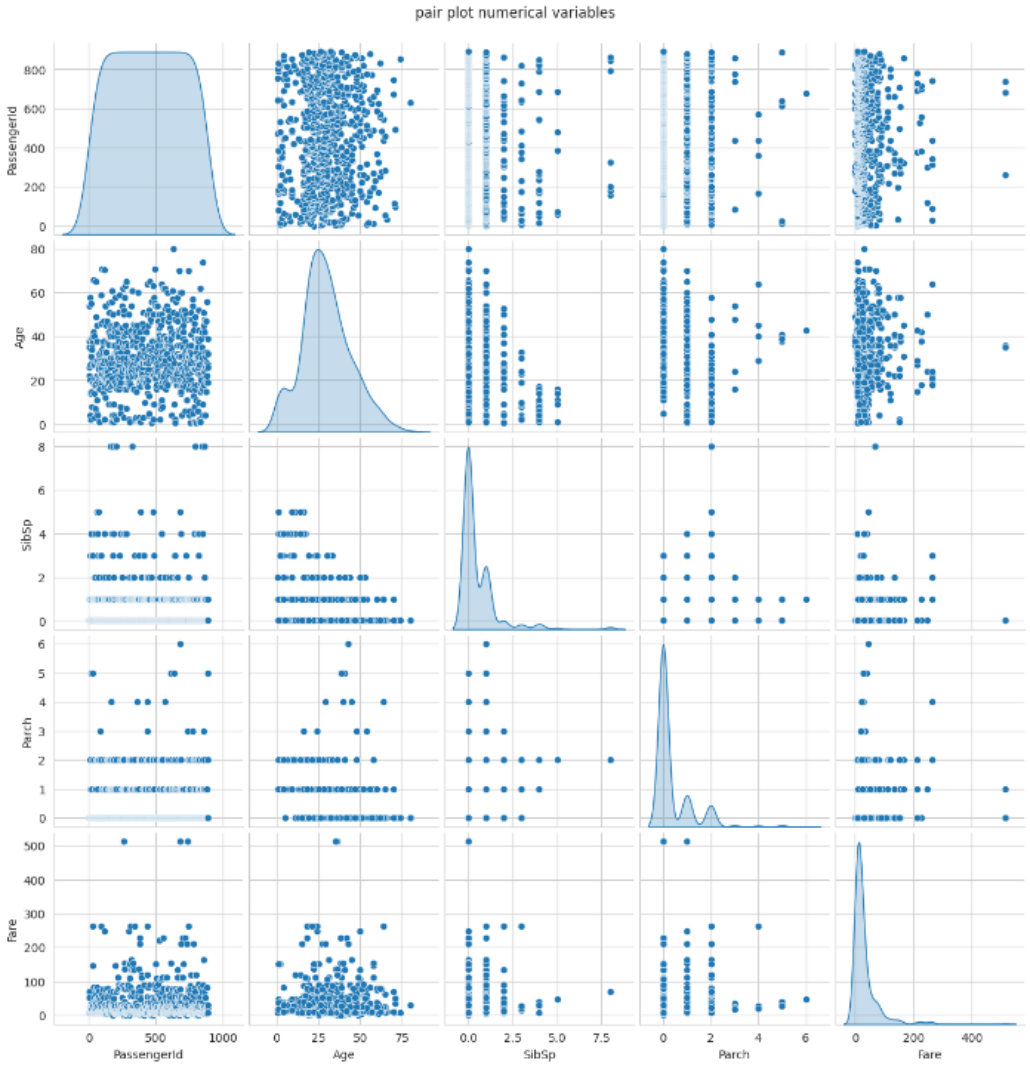
Titanic - Machine Learning from Disaster
Start here! Predict survival on the Titanic and get familiar with ML basics
Getting Started · 15752 Teams · Ongoing

557/15752

1 Analyse des données et prétraitement

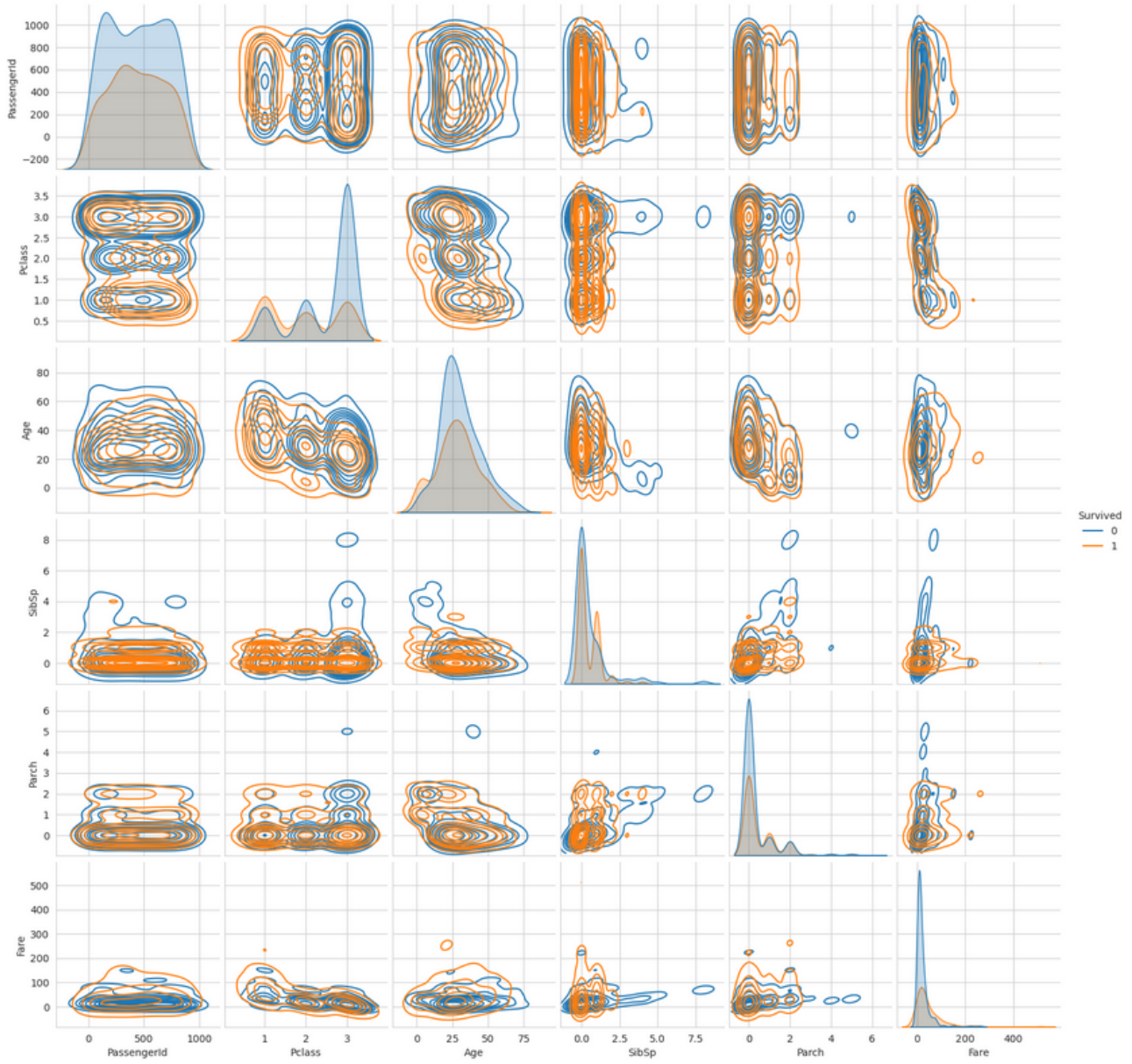


J'ai commencé par réaliser un graphique simpliste, permettant de se concentrer sur l'essentiel. Chaque barre représente une tranche d'âge de 10 ans et la hauteur de la barre indique le pourcentage de passagers qui se situent dans cette tranche d'âge. Les valeurs null ont été exclues. Plus un décile est représenté dans les données, moins la couleur de celle-ci est opaque. Nous permettant de voir dès le premier coup d'oeil que la plage 20/30 ans est la plus représentée.



J'ai ensuite réalisé une matrice de dispersion des variables quantitatives du jeu de données, m'ayant permis de remarquer les points suivants:

- Beaucoup de jeunes passagers.
- Très peu de passagers avec des frères et soeurs ou des conjoints.
- Peu de passagers avec leurs parents ou enfants.
- La majorité des billets vendus étaient à bas prix.
- Il semblerait que les billets aux prix les plus élevés, ont été vendus aux personnes plus agées en moyenne.



J'ai ensuite réalisé une matrice de dispersion colorée par la variable catégorielle Survided, m'ayant permis de remarquer les points suivants:

- Les passagers de première classe avaient de meilleurs chances de survie.
- D'avantage de suivants parmis les enfants.
- Les familles de 2 ou 3 personnes ont plus de chances de survie que les familles plus nombreuses ou les personnes seules.

Le code ci-dessous, dédié au prétraitement des données réalise les étapes suivantes:

- Normalisation des noms (retire les points, sépare les noms par des espaces).
- Extraction du numéro de billet et séparation des élémentrs, retourne NONE si un seul élément est trouvé.
- Utilisation de Tensorflow pour la tokenization.
- Conversion des dataframes en datasets TensorFlow avec la colonne Survived comme label.
- Suppresion de certaines colonnes pour ne pas les utiliser comme caractéristiques d'apprentissage du modèle(PassengerId car inutile et Survided car sert déjà de label).

```
def preprocess(df):
df = df.copy()

def normalize_name(x):
    return " ".join([v.strip(",()[].\"'") for v in x.split(" ")])

def ticket_number(x):
    return x.split(" ")[-1]

def ticket_item(x):
    items = x.split(" ")
    if len(items) == 1:
        return "NONE"
    return "_".join(items[:-1])

df["Name"] = df["Name"].apply(normalize_name)
df["Ticket_number"] = df["Ticket"].apply(ticket_number)
df["Ticket_item"] = df["Ticket"].apply(ticket_item)
return df

def tokenize_names(features, labels=None):
features["Name"] = tf.strings.split(features["Name"])
return features, labels

preprocessed_train_df = preprocess(train_df)
preprocessed_test_df = preprocess(test_df)

input_features = list(preprocessed_train_df.columns)
input_features.remove("Ticket")
input_features.remove("PassengerId")
input_features.remove("Survived")

train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(preprocessed_train_df, label="
    Survived").map(tokenize_names)
serving_ds = tfdf.keras.pd_dataframe_to_tf_dataset(preprocessed_test_df).map(
    tokenize_names)
```

Listing 1: Traitement des données

2 Choix du modèle et analyse des résultats

```
def prediction_to_kaggle_format(model, threshold=0.5):
proba_survive = model.predict(serving_ds, verbose=0)[:, 0]
return pd.DataFrame({
    "PassengerId": preprocessed_test_df["PassengerId"],
    "Survived": (proba_survive >= threshold).astype(int)
})

num_trees = 300
min_examples = 5
categorical_algorithm = "CART"

model = tfdf.keras.GradientBoostedTreesModel(
    verbose=0,
    features=[tfdf.keras.FeatureUsage(name=n) for n in input_features],
    exclude_non_specified_features=True,
    random_seed=1234,
    num_trees=num_trees,
    min_examples=min_examples,
    categorical_algorithm=categorical_algorithm
)
model.fit(train_ds)
```

Listing 2: Traitement des données

Le code ci-dessus présente l'utilisation du modèle GBT, pour Gradient Boosted Trees, un modèle puissant et robuste face aux différentes échelles et types de données. Le nombre d'arbres est de 300, plus il y a d'arbres, plus le modèle est puissant. Cependant, plus le nombre d'arbres augmente, plus le modèle risque de trop s'ajuster aux données. En voulant augmenter le nombre d'arbres, j'ai fais passer mon score Kaggle de 0.80382 à moins de 0.7...
J'ai utilisé l'algorithme CART pour sa grand flexibilité, ses performances en classification et régression, ainsi que sa facilité d'interprétation. J'ai utilisé pour mon premier script "dumb", un modèle random forest assez basique, avec les bons réglages j'ai pu obtenir un score de 0.75 sur Kaggle, me plaçant dans les 12 000 premiers. Une fois ce modèle testé, j'ai regardé ce qu'il été commun de faire pour s'améliorer dans ce challenge, je me suis aperçu que des arbres de décisions ou des GBT permettaient d'obtenir de très bons scores (entre 0.8 et 0.9). J'ai en effet obtenu mes meilleurs résultats avec GBT, dans mon cas, meme XGB n'a pas su faire mieux...
Je suis persuadé qu'avec un meilleur prétraitement des données, comme dans cet exemple (https://www.kaggle.com/code/vbmokin/titanic-top-3-cluster-analysis/notebook), et le modèle GBT, nous pourrions faire un meilleur score