notebook

June 28, 2026

1 Titanic Kaggle Competition

V0 goal: Set a baseline for modeling performance metrics. Meaning, run the data through a few models with as little feature engineering as possible. Use only single algorithms - no ensembles.

1.1 Import and Config

[19]:
```
# import necessary packages and configurations

from pathlib import Path
import yaml
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

# Load configuration from YAML file

NOTEBOOK_DIR = Path.cwd()
PROJECT_DIR = NOTEBOOK_DIR.parents[1]

# print(f"Notebook directory: \n{NOTEBOOK_DIR}")
# print(f"Project directory: \n{PROJECT_DIR}")

# Explicitly read, load and close the config.yaml file

config_path = PROJECT_DIR / "config.yaml"
file = config_path.open("r")
config = yaml.safe_load(file)
file.close()

# print(f"\nConfiguration loaded from: \n{config_path}:")
# print(json.dumps(config, indent=3))
```

1.2 Breaking down the problem

We know we need to decide survival which is a value of Yes or No (1 || 0).

Is this a classification problem? Yes, specifically it is a *binary classification*.

What are Binary Classification algorithms? 1. Logistic Regression 1. K-Nearest Neighbors 1. Naive Bayes 1. Decision Tree Classifier 1. Random Forest Classifier

1.3 Preprocessing

[20]:
```
df = pd.read_csv(PROJECT_DIR / "data" / "raw" / "train.csv")
#display(df.head(5))
```

Feature engineering for modeling compatibility. Mostly means filling NaN values and converting string values.

[21]:
```
nan_cols = []
for col in df.columns:
    if df[col].isnull().any():
        nan_cols.append(col)

print("Columns with NaNs:", nan_cols)
```

```
Columns with NaNs: ['Age', 'Cabin', 'Embarked']
```

Age, Cabin and Embarked must be adjusted for NaN values.

Create a re-usable function for preprocessing to make sure preprocessing for model training is the same for competition prediction data.

[22]:
```
def preprocess_data(df):
    df = df.copy()
    df = pd.get_dummies(df, columns=["Sex", "Embarked"])
    df["Age"] = df["Age"].fillna(df["Age"].median())
    df = df.drop(columns=["Name", "Fare", "Ticket", "Cabin"], errors="ignore")
    return df
```

[23]:
```
df = preprocess_data(df)

# drop y variable for X dataset
X = df.drop(columns=["Survived"])
display(X)

# setup the Y variable
y = df["Survived"]
display(y)
```

```
     PassengerId  Pclass   Age  SibSp  Parch  Sex_female  Sex_male  \
0              1       3  22.0      1      0       False      True
1              2       1  38.0      1      0        True     False
2              3       3  26.0      0      0        True     False
```

```
3              4       1  35.0      1      0        True     False
4              5       3  35.0      0      0       False      True
..           …     …   …    …    …         …       …
886          887       2  27.0      0      0       False      True
887          888       1  19.0      0      0        True     False
888          889       3  28.0      1      2        True     False
889          890       1  26.0      0      0       False      True
890          891       3  32.0      0      0       False      True

     Embarked_C  Embarked_Q  Embarked_S
0         False       False        True
1          True       False       False
2         False       False        True
3         False       False        True
4         False       False        True
..          …         …         …
886       False       False        True
887       False       False        True
888       False       False        True
889        True       False       False
890       False        True       False

[891 rows x 10 columns]
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

1.4 Model Training

Split the X and y data into train and test datasets.

[24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=18,␣
↪test_size=0.2)
```

This is the data which the model is trained on:

[25]:
```
display(X_train.describe().round(2))
```

```
       PassengerId  Pclass     Age   SibSp   Parch
count       712.00  712.00  712.00  712.00  712.00
```

```
mean        451.82    2.32   29.57    0.53    0.37
std         258.56    0.84   12.57    1.15    0.78
min           2.00    1.00    0.42    0.00    0.00
25%         226.75    2.00   23.00    0.00    0.00
50%         449.50    3.00   28.00    0.00    0.00
75%         674.25    3.00   35.00    1.00    0.00
max         891.00    3.00   80.00    8.00    5.00
```

Below is a preview of the subset of data we will use to test the model's performance.

[26]:
```
display(X_test.describe().round(2))
```

```
       PassengerId  Pclass     Age   SibSp   Parch
count       179.00  179.00  179.00  179.00  179.00
mean        422.85    2.25   28.54    0.49    0.42
std         251.90    0.83   14.69    0.87    0.89
min           1.00    1.00    0.67    0.00    0.00
25%         197.00    1.50   19.50    0.00    0.00
50%         425.00    3.00   28.00    0.00    0.00
75%         636.50    3.00   35.50    1.00    1.00
max         878.00    3.00   74.00    4.00    6.00
```

1.4.1 Logistic Regression Classifier

Choose the model and configure parameters. With this being a baseline, I had to give it more iterations to start. This is because different features might have different magnitudes which can be addressed by things like Scalars.

[27]:
```
model = LogisticRegression(max_iter=100000)

logit_model = model.fit(X_train, y_train)

logit_predictions = logit_model.predict(X_test)

logit_score = logit_model.score(X_test, y_test)

print(logit_score)
```

```
0.8603351955307262
```

1.4.2 Nearest Neighbor Classifier

[28]:
```
model = KNeighborsClassifier()

NN_model = model.fit(X_train, y_train)

NN_Predictions = NN_model.predict(X_test)

NN_score = NN_model.score(X_test, y_test)
```

```
print(NN_score)
```

```
0.5195530726256983
```

1.4.3 Gaussian Naive Bayes Classifier

[29]:
```
model = GaussianNB()

GNB_model = model.fit(X_train, y_train)

GNB_Predictions = GNB_model.predict(X_test)

gnb_score = GNB_model.score(X_test, y_test)

print(gnb_score)
```

```
0.8379888268156425
```

1.4.4 Decision Tree Classifier

[30]:
```
model = DecisionTreeClassifier()

DT_model = model.fit(X_train, y_train)

DT_Predictions = DT_model.predict(X_test)

dt_score = DT_model.score(X_test, y_test)

print(dt_score)
```

```
0.7877094972067039
```

1.4.5 Random Forest Classifier

[31]:
```
model = RandomForestClassifier(n_estimators=100)

RF_model = model.fit(X_train, y_train)

RF_Predictions = RF_model.predict(X_test)

rf_score = RF_model.score(X_test, y_test)

print(rf_score)
```

```
0.8379888268156425
```

1.5 Analyze Model Performances

[32]:
```
all_scores = {"Logistic Regression":logit_score,"Nearest Neighbor":
 ↪NN_score,"Guassian Naive Bayes":gnb_score,"Decision Tree":dt_score,"Random␣
 ↪Forest":rf_score}
display(all_scores)
```

```
{'Logistic Regression': 0.8603351955307262,
 'Nearest Neighbor': 0.5195530726256983,
 'Guassian Naive Bayes': 0.8379888268156425,
 'Decision Tree': 0.7877094972067039,
 'Random Forest': 0.8379888268156425}
```

Take the highest score for all model accuracy scores and save to the Kaggle desired format for competition submission.

[33]:
```
models = {
    "Logistic Regression": logit_model,
    "Nearest Neighbor": NN_model,
    "Gaussian Naive Bayes": GNB_model,
    "Decision Tree": DT_model,
    "Random Forest": RF_model
}

best_model_name = max(all_scores, key=all_scores.get)
best_model = models[best_model_name]
best_model_score = all_scores[best_model_name]

print(best_model_name)
print(best_model_score)
print(best_model)
```

```
Logistic Regression
0.8603351955307262
LogisticRegression(max_iter=100000)
```

1.6 Submit Competition Predictions

Preprocess the competition data with the encapsulated preprocessing function made earlier. It is important to be sure the preprocessing for the Kaggle submission data set is the same as the model which was trained.

[34]:
```
comp_df = pd.read_csv(PROJECT_DIR / "data" / "raw" / "test.csv")

comp_df = preprocess_data(comp_df)

comp_predictions = best_model.predict(comp_df)
```

Format predictions.

Kaggle confirms the submitted csv should be 418 rows and a header.

[35]:
```
submission = pd.DataFrame({
    "PassengerId": comp_df["PassengerId"],
    "Survived": comp_predictions
})

print(submission.shape)
```

```
(418, 2)
```

Create submission file.

This is what is submitted to Kaggle and the predictions are scored against the actual hidden outcomes.

[36]:
```
submission.to_csv("Titanic_V0_Kaggle_Submission.csv", index=False)
print("Submission file generated & saved.")
```

```
Submission file generated & saved.
```