Data Mining- Term project

Apoorva Bandi

Titanic - Machine Learning from Disaster

Description of the project:

We develop a model that predicts which passengers of the Titanic shipwreck survived with the help of machine learning.

There are data three files for the project:

Train.csv: Train.csv is a spreadsheet that includes information about a portion of the passengers on board; specifically, it includes 891 passengers, each of whom has a separate row in the table. To ascertain whether or not each passenger survived, utilize the values in the second column ("Survived"):

"1" indicates that the passenger survived.

The passenger died, if the number is "0".

Test.csv: We must determine whether the remaining 418 passengers (on test.csv) on board survived by using the patterns we discover in train.csv.

It should be observed that test.csv does not have a "Survived" column because this information is withheld . The performance in the competition will be based on how well we can predict these hidden values.

Gender_submission.csv: There is a gender_submission.csv file as an example of how the predictions should be organized. According to the prediction, every male passenger died and every female passenger lived. We will most likely have different survival theories, which will result in a different submission file. Our submission ought to have:

A "PassengerId" column with each test passenger's ID in it.csv, a "Survived" column where you mark rows where you believe the passenger to have survived with a "1" and rows where you believe the passenger to have died with a "0".

Now, we will develop our own machine learning model to improve our predictions of whether a passenger survived or not.

Implementation:

1.1. Importing libraries:

Here we import the ‘numpy’ and ‘pandas’ libraries for data analysis and numerical operations. The code ran successfully and three lines of output are shown. This means our outputs are saved at that particular location.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/titanic/train.csv
/kaggle/input/titanic/test.csv
/kaggle/input/titanic/gender_submission.csv
```

1.2. Load the data

```
train_data = pd.read_csv("/kaggle/input/titanic/train.csv")
train_data.head(5)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Here, we plug in to the location of the training data file and then we check whether it is properly located or not. The .head(5) operation gives us the top 5 rows of the training data. The code is successfully implemented as we got the output. Same goes for the testing data. We plug in to the location of the testing data and check whether it is properly located or not. The .head(5) operation gives the top 5 rows of the testing data.

```
test_data = pd.read_csv("/kaggle/input/titanic/test.csv")
test_data.head(5)
```

[3]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

1.3. Cleaning the data, Exploring the pattern

```
test_data = pd.read_csv("/kaggle/input/titanic/test.csv")
def clean(train_data):
    train_data = train_data.drop(["Ticket","Cabin","Name"],axis = 1)

    fcols = ["SibSp","Parch","Fare","Age"]
    for col in fcols:
        train_data[col].fillna(train_data[col].median(),inplace=True)
    train_data.Embarked.fillna("U",inplace = True)
    return train_data
train_data = clean(train_data)
test_data = clean(test_data)
```

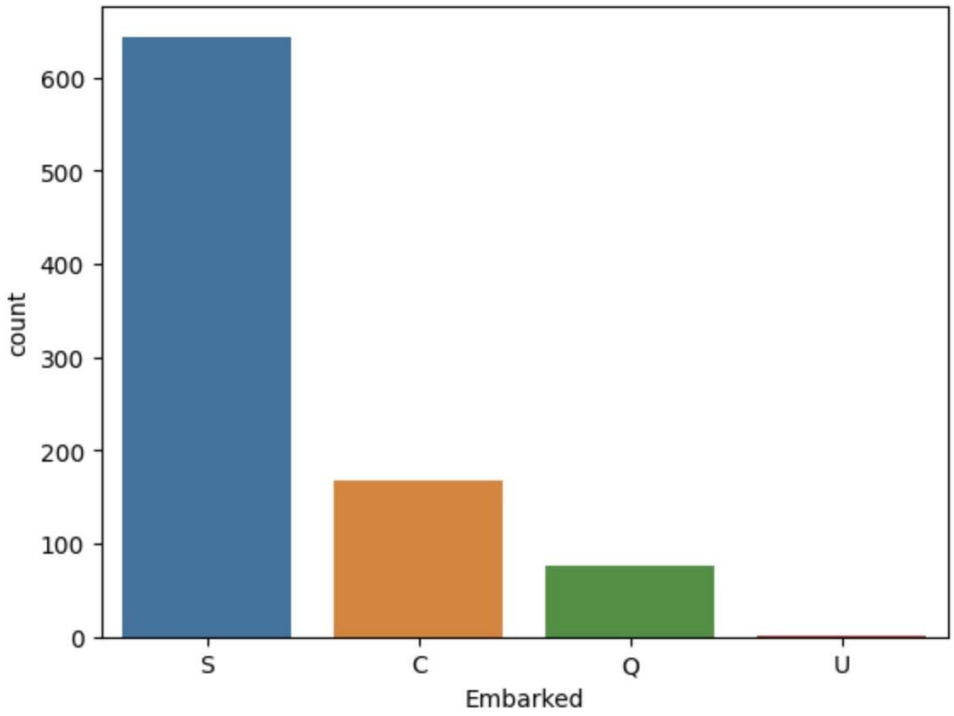
+ Code + Markdown

The above function cleans the data and removes the columns which we would not be using for the classification and prediction later in the code. And for the columns "SibSp", "Parch", "Fare", "Age" we fill in the null values by taking the median for all the values. Now, the data is clean.

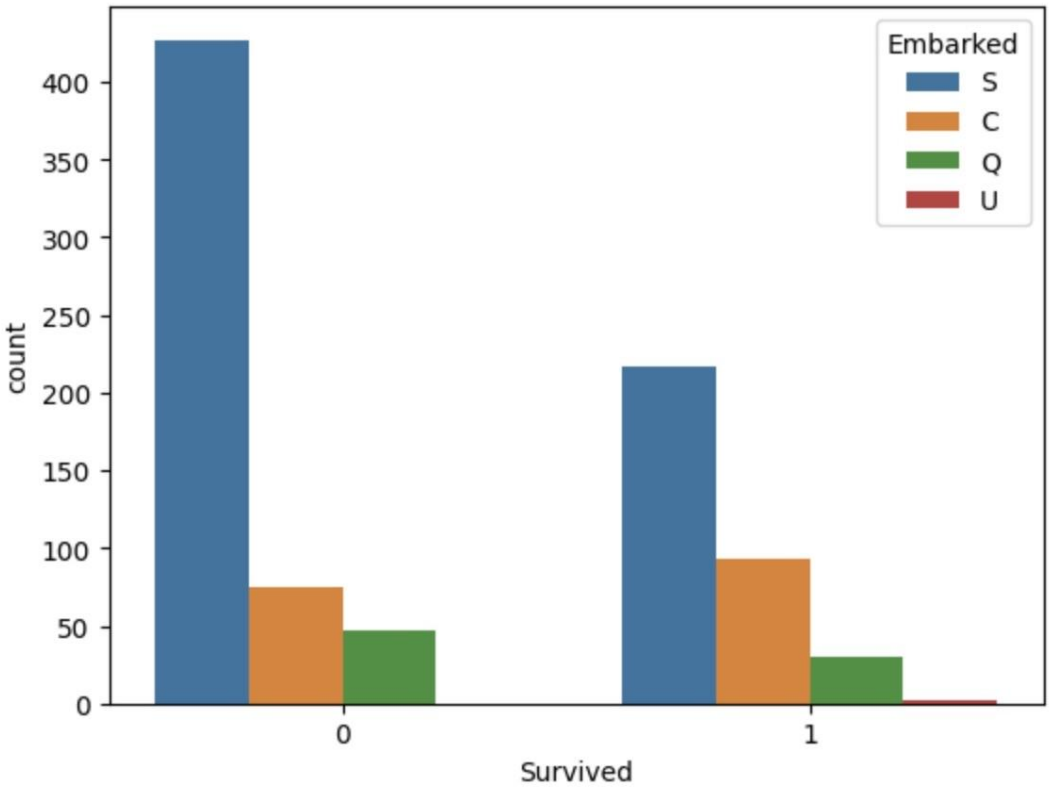
1.4. Data Visualization

Here, we use matplotlib and seaborn libraries to plot the data of Embarked and Survived to look at the differences based on it, which can help us make informed decisions as to which columns to choose in the classification process.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(train_data, hue= 'Embarked',x= 'Survived')
plt.show()
sns.countplot(train_data, x = 'Embarked')
plt.show()
```



+ Code + Markdown



By these graphs, we will be able to visualize the data and analyze to properly predict the survival passengers.

1.5. Preprocessing the data

```
train_data.head(5)
```

[20…

	PassengerId	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

+ Code + Markdown

As we dropped the columns and changed them , we now have the clean data.

Now, we preprocess the data by label encoding. We fit and transform the data at the same time. This will help in doing the mapping of the categories. The classes will be converted to the integers.

```
from sklearn import preprocessing
label= preprocessing.LabelEncoder()

lcols = ["Sex","Embarked"]

for col in lcols:
    train_data[col] = label.fit_transform(train_data[col])
    test_data[col] = label.transform(test_data[col])
    print(label.classes_)

train_data.head(5)
```

```
['female' 'male']
['C' 'Q' 'S' 'U']
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	2
1	2	1	1	0	38.0	1	0	71.2833	0
2	3	1	3	0	26.0	0	0	7.9250	2
3	4	1	1	0	35.0	1	0	53.1000	2
4	5	0	3	1	35.0	0	0	8.0500	2

+ Code + Markdown

1.6. Machine Learning Model

```
from sklearn.ensemble import RandomForestClassifier

y = train_data["Survived"]

features = ["Pclass", "Sex", "SibSp", "Parch"]
X = pd.get_dummies(train_data[features])
X_test = pd.get_dummies(test_data[features])

model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=1)
model.fit(X, y)
predictions = model.predict(X_test)

output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
output.to_csv('submission.csv', index=False)
print("Your submission was successfully saved!")
```

```
Your submission was successfully saved!
```

In the code cell below, patterns are searched across four distinct data columns ("Pclass", "Sex", "SibSp", and "Parch"). Based on patterns found in the train, it builds the trees in the random forest model.csv file, prior to producing forecasts for the test passengers. These updated forecasts are also saved by the code and submitted as a CSV file.csv. The output displays as “your submission was successfully saved”. The accuracy percentage is 0.7751%.

My Contribution:

```
import pandas as pd
from sklearn import svm

def svm_classifier():
    features = ["Pclass","Sex","SibSp","Parch"]
    X_train = pd.get_dummies(train_data[features])
    y_test = train_data["Survived"]

    svm_model = svm.SVC(kernel='poly', degree=3, C=1, random_state=42)
    svm_model.fit(X_train,y_test)
    print("Model Score:",svm_model.score(X_train, y_test))

    X_test = pd.get_dummies(test_data[features])
    predictions = svm_model.predict(X_test)

    output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
    output.to_csv('submission.csv', index=False)
    print("Your submission was successfully saved!")
svm_classifier()
```

```
Model Score: 0.8058361391694725
Your submission was successfully saved!
```

Here, I have used the SVM classification model.First, I have preprocessed and cleaned the training and testing data by filling the null values and dropping the unnecessary columns. Next, the svm_classifier function prepares the features for the training inputs. We create the SVM model based on the four data columns ("Pclass", "Sex", "SibSp", and "Parch"). Here, classification is accomplished through the use of a Support Vector Classifier (SVC). SVC maps data points to a high-dimensional space and then selects the best hyperplane to split the data into two classes, which are survived or not. Here the dataset is used to fit into the model.

By using the model score function we will be able to see the accuracy of the model. Next, the prediction is done by comparing the submission file whether the passenger survived or not. After the implementation we will be able to see the submission is successfully saved.

By using the SVM classifier, I was able to get the accuracy as 0.7775%.