Assignment 1

Titanic- Machine Learning from Disaster

Hongyi Hao

1 Problem Description

1.1 Problem 1:

Which feature do you believe is the most important for the model's performance? Justify your answer with evidence, such as data analysis, visualizations, or feature importance scores.

1.2 Problem 2:

For the "Age" feature, what alternative methods could be used to handle missing values instead of filling them with 0? Explain how these methods could improve the model's performance.

2 Data Preparation

2.1 Read and Visualize Data

(a) Read the train.csv file.

(b) Print out the first 5 samples.

To begin with, I used the ***pandas.read_csv*** function in Python to read the train.csv data. And then print out the first 5 samples.

```
df = pd.read_csv('train.csv')
print("First 5 rows of data:")
print(df.head())
```

Output:

```
First 5 rows of data:
   PassengerId  Survived  Pclass                                                Name     Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                             Braund, Mr. Owen Harris    male  ...      0         A/5 21171   7.2500   NaN         S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  ...      0          PC 17599  71.2833   C85         C
2            3         1       3                              Heikkinen, Miss. Laina  female  ...      0  STON/O2. 3101282   7.9250   NaN         S
3            4         1       1        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  ...      0            113803  53.1000  C123         S
4            5         0       3                            Allen, Mr. William Henry    male  ...      0            373450   8.0500   NaN         S
```

2.2 Handling Missing Data

(a) Print out the count of missing values in each column.

(b) Impute missing entries with zero.

Then, I used the ***pandas.DataFrame.count*** and ***pandas.DataFrame.fillna*** function to count the number of values in each column and impute missing entries with 0.

```
print("\nMissing values count per column (using DataFrame.count()):")
total_rows = len(df)
non_null_counts = df.count()
missing_counts = total_rows - non_null_counts
print(missing_counts)

df = df.fillna(0)
```

Output:

```
Missing values count per column (using DataFrame.count()):
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Comparison Before And After Filling:

```
PS C:\Users\Lanyi\Desktop\Project\Distributed_Machine_Learning_Foundations_and_Algorithms\Assignment\Assignment_1\Week_1_titanic> & D:/anaconda3/python.exe c:/Users/Lanyi/Desktop/Project/Distributed_Machine_Learning_Foundations_and_Algorithms/Assignment/Assignment_1/Week_1_titanic/test.py

Missing values(Before):
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

Missing values(After):
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

2.3 Handling numerical and categorical values.

(a) Identify and write down which columns are numerical and which are categori cal.

(b) Convert the categorical columns into one-hot encoded columns. Print out the first 5 samples to verify.

Additionally, I used the ***pandas.DataFrame.head*** function and ***pandas.get_dummies*** function to Identify and write down the type of the values convert the categorical columns into one-hot encoded columns.

```
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

print("Numerical Columns:")
print(numerical_columns)

print("\nCategorical Columns:")
print(categorical_columns)
```

```
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

print("\nFirst 5 rows after one-hot encoding:")
print(df_encoded.head())
```

Output:

```
Numerical Columns:
['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']

Categorical Columns:
['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']
```

```
First 5 rows after one-hot encoding:
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  ...  Cabin_F38  Cabin_F4  Cabin_G6  Cabin_T  Embarked_C  Embarked_Q  Embarked_S
0            1         0       3  22.0      1      0   7.2500  ...      False     False     False    False       False       False        True
1            2         1       1  38.0      1      0  71.2833  ...      False     False     False    False        True       False       False
2            3         1       3  26.0      0      0   7.9250  ...      False     False     False    False       False       False        True
3            4         1       1  35.0      1      0  53.1000  ...      False     False     False    False       False       False        True
4            5         0       3  35.0      0      0   8.0500  ...      False     False     False    False       False       False        True

[5 rows x 1728 columns]
```

2.4 Dataset Splitting

(a) Divide the cleaned dataset into training features (x_train), training targets (y train), testing features (x_test), and test targets (y_test).

(b) Print out each set's shape.

```
X = df.drop("Survived", axis=1)
y = df["Survived"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("\nShapes:")
print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
print("y_train:", y_train.shape)
print("y_test:", y_test.shape)
```

Output:

```
Shapes:
X_train: (712, 1732)
X_test: (179, 1732)
y_train: (712,)
y_test: (179,)
```

2.5 Addressing Imbalanced Labels for Training Set

(a) How many samples are there for survival and non-survival?

(b) Use oversampling technique to handle imbalanced classes.

```
print("\nClass distribution before:")
print(y_train.value_counts())
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
print("Class distribution after:")
print(y_resampled.value_counts())
```

Output:

```
Class distribution before:
Survived
0    444
1    268
Name: count, dtype: int64
Class distribution after:
Survived
0    444
1    444
Name: count, dtype: int64
```

3 Model Training and Validation

3.1 Training a k-nearest neighbors classifier using the training dataset you have from prev section.

3.2 Print out your model's prediction accuracy on the training dataset.

3.3 Print out your model's prediction accuracy on the test dataset

3.4 Plot the training/validation accuracy with respect to different values of k. The range of k should be [1, 100]. Save the plotted to your pdf submission file.

```
def train_knn(X_train, y_train, X_test, y_test):
    accuracies_train = []
    accuracies_test = []
    k_range = range(1, 101)
    print("KNN Start...\n")
    print(f"{'k':<5} {'Train Acc':<12} {'Test Acc':<12}")
    print("-" * 30)

    for k in k_range:
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X_train, y_train)

        y_pred_train = model.predict(X_train)
        y_pred_test = model.predict(X_test)
```

```
15
16          acc_train = accuracy_score(y_train, y_pred_train)
17          acc_test = accuracy_score(y_test, y_pred_test)
18
19          accuracies_train.append(acc_train)
20          accuracies_test.append(acc_test)
21
22          print(f"{k:<5} {acc_train:<12.4f} {acc_test:<12.4f}")
23
24      # Find the best k
25      best_k = accuracies_test.index(max(accuracies_test)) + 1
26      best_train_acc = accuracies_train[best_k - 1]
27      best_test_acc = accuracies_test[best_k - 1]
28
29      # Paint the graph
30      plt.figure(figsize=(10,6))
31      plt.plot(k_range, accuracies_train, label='Train Accuracy')
32      plt.plot(k_range, accuracies_test, label='Test Accuracy')
33      plt.xlabel("k")
34      plt.ylabel("Accuracy")
35      plt.title("KNN Accuracy vs k")
36      plt.legend()
37      plt.grid()
38      plt.savefig("screenshots/knn_accuracy.png")
39      plt.show()
40
41      print(f"\n Best test accuracy: {best_test_acc:.4f} at k =
    {best_k}")
42      print(f" Corresponding train accuracy: {best_train_acc:.4f}")
43
44      print("\n Evaluating model using best k:")
45      best_model = KNeighborsClassifier(n_neighbors=best_k)
46      best_model.fit(X_train, y_train)
47
48      final_train_pred = best_model.predict(X_train)
49      final_test_pred = best_model.predict(X_test)
50
51      final_train_acc = accuracy_score(y_train, final_train_pred)
52      final_test_acc = accuracy_score(y_test, final_test_pred)
53
54      print(f" Final train accuracy with k = {best_k}:
    {final_train_acc:.4f}")
55      print(f" Final test accuracy with k = {best_k}:
    {final_test_acc:.4f}")
```

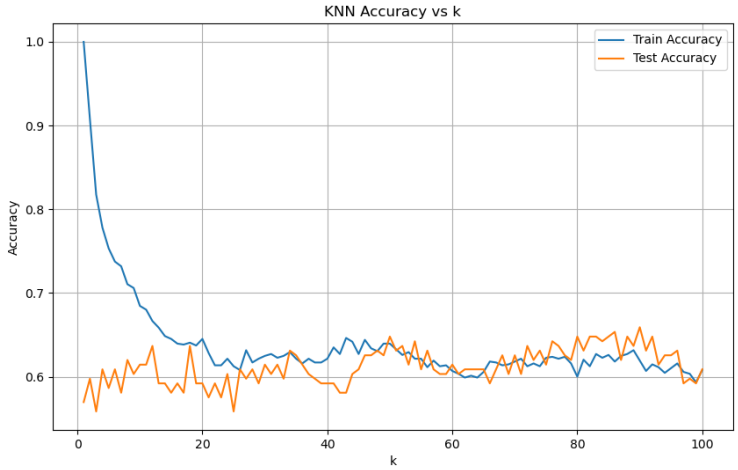
According to the test and experiment the model's prediction accuracy on the training dataset and the model's prediction accuracy on the test dataset are as followed:

```
Best test accuracy: 0.6592 at k = 90
Corresponding train accuracy: 0.6194
```

All in all, evaluating model using best k:

```
Evaluating model using best k:
Final train accuracy with k = 90: 0.6194
Final test accuracy with k = 90: 0.6592
```

And then, the training/validation accuracy with respect to different values of k. The range of k should be [1, 100].



4 Analytical Questions

1. Which feature do you believe is the most important for the model's performance?

In my opinion, the most important factor affecting model performance is 'Sex', as analyzed using Python based on the original dataset:

```python
import pandas as pd
df = pd.read_csv("train.csv")
survival_rate_by_sex = df.groupby("Sex")["Survived"].mean()
print("Average survival rate by sex:")
print(survival_rate_by_sex)
```

Output:

```
Average survival rate by sex:
Sex
female    0.742038
male      0.188908
```

The output shows a significant disparity in survival rates between passengers of different genders, indicating that the survival rate for women was much higher than that for men. This also reflects the prevailing rescue strategy of 'women and children first', which contributed to the higher survival rate of women in the Titanic disaster. At the same time, After OneHot encoding, the discriminative ability of the KNN model can be significantly improved, and compared to other high-dimensional sparse features, the information density is higher and the effect is more clear.

2. For the "Age" feature, what alternative methods could be used to handle missing values instead of filling them with 0?

When dealing with missing values for the 'Age' feature, directly filling them with 0 can introduce outliers and potentially mislead the model's judgments, which may affect subsequent model training. There are 4 other methods to deal with this feature.

(1) Fill with median:

Use the median of the Age from the entire dataset to fill in missing values, which reduces the impact of outliers. The median is more robust and can alleviate the disturbance to the model caused by filling in 0 or the mean value.

```
df['Age'] = df['Age'].fillna(df['Age'].median())
```

(2) Fill in using the median grouped by 'Pclass':

Fill in the missing ages by calculating the median age for each group based on passengers' Pclass. This method takes advantage of the potential correlation between 'cabin class and age', filling in based on categorical features, which is more in line with the true data distribution.

```
df['Age'] = df.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.median()))
```

(3) Using regression models to predict missing values:

Use known features such as “Pclass”, Sex, and Fare to train a regression model to predict Age. This is a machine learning method based on the relationships between other features, which is more flexible than simple statistics and can provide personalized fill values, improving the quality of the imputation.

```python
from sklearn.ensemble import RandomForestRegressor

age_df = df[['Age', 'Pclass', 'Sex', 'SibSp', 'Parch', 'Fare']]
age_df = pd.get_dummies(age_df)

known_age = age_df[age_df['Age'].notnull()]
unknown_age = age_df[age_df['Age'].isnull()]

X_train = known_age.drop('Age', axis=1)
y_train = known_age['Age']
X_pred = unknown_age.drop('Age', axis=1)

rfr = RandomForestRegressor(random_state=0, n_estimators=100)
rfr.fit(X_train, y_train)
df.loc[df['Age'].isnull(), 'Age'] = rfr.predict(X_pred)
```

(4) Add 'Is Age Missing' as a new feature:

While filling in Age = 0, add a new feature "Age_missing" to indicate whether it is missing. This approach explicitly informs the model that 'this is missing data,' allowing the model to learn whether 'missingness' itself is related to the prediction target (Survived).

```python
df['Age_missing'] = df['Age'].isnull().astype(int)
```

Method	median	grouped by 'Pclass'	regression models	Add Age Missing
Accuracy	0.5866	0.6034	0.6201	0.5866

Form 1: Accuracy Comparison

More reasonable methods for handling missing values, such as filling with group medians, regression prediction, and missing indicator features, are not only more in line with the data distribution, but also significantly enhance the model's ability to learn real situations, thereby improving classification performance. Especially when using regression prediction for filling, the accuracy compared to other methods and the original method shows a significant improvement in the final accuracy.