

```
import numpy as np
import pandas as pd
import sklearn as sk
```

```
# Read files
train_data = pd.read_csv("train.csv")
test_df = pd.read_csv('test.csv')
```

```
# Drop unneeded columns
cols_to_drop = ['Cabin', 'Name', 'Ticket']
train_data = train_data.drop(columns=cols_to_drop)
test_df = test_df.drop(columns=cols_to_drop)
train_data = train_data.drop("PassengerId", axis=1)
```

```
# One hot encode
def one_hot_encode_sex_embarked(df):
    # Remove rows where Embarked is NaN
    df = df.dropna(subset=['Embarked'])
    # One hot encode 'Sex' and 'Embarked'
    df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=False)
    return df
```

```
# Fill missing ages with median age
def fill_missing_ages(df):
    median_age = df['Age'].median()
    df['Age'] = df['Age'].fillna(median_age)
    return df
```

```
# Normalize features
def normalize_features(df, minmax_cols, standard_cols, fit_stats=None):
    df_norm = df.copy()
    if fit_stats is None:
        fit_stats = {}
        for col in minmax_cols:
            fit_stats[col] = {
                'min': df_norm[col].min(),
                'max': df_norm[col].max()
            }
        for col in standard_cols:
            fit_stats[col] = {
                'mean': df_norm[col].mean(),
                'std': df_norm[col].std()
            }
    for col in minmax_cols:
        min_val = fit_stats[col]['min']
        max_val = fit_stats[col]['max']
        df_norm[col] = (df_norm[col] - min_val) / (max_val - min_val)
    for col in standard_cols:
        mean = fit_stats[col]['mean']
        std = fit_stats[col]['std']
        df_norm[col] = (df_norm[col] - mean) / std
    if fit_stats is not None:
        return df_norm, fit_stats
    else:
        return df_norm
```

```
# One hot encode and fill missing ages
train_data = one_hot_encode_sex_embarked(train_data)
test_df = one_hot_encode_sex_embarked(test_df)
train_data = fill_missing_ages(train_data)
test_df = fill_missing_ages(test_df)
```

```
# Normalize features
minmax_cols = ['Pclass', 'Fare', 'Age', 'SibSp', 'Parch']
standard_cols = []
fit_stats = {}
train_data, fit_stats = normalize_features(train_data, minmax_cols, standard_cols)
test_df, _ = normalize_features(test_df, minmax_cols, standard_cols, fit_stats)
```

```
# sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

```
# Initialize random params function
```

```
def initialize_params(n_features, seed=39):
    np.random.seed(seed)
    w = np.random.randn(n_features, 1) * 0.01
    b = 0.0
    return w, b
```

```
# Sigmoid
```

```
def forward(X, w, b):
    m_samples = X.shape[0]
    Z = np.dot(X, w) + b
    a = sigmoid(Z)
    return a
```

```
# Compute cost
```

```
def compute_cost(y, y_hat, w):
    m_samples = y.shape[0]
    epsilon = 1e-15
    cost = -(1 / m_samples) * np.sum((y * np.log(y_hat + epsilon) + (1 - y) * np.log(1 - y_hat + epsilon)))
    return cost
```

```
# Compute gradients
```

```
def compute_gradients(X, y, y_hat, w, reg_lambda=0.0):
    m_samples = X.shape[0]
    dw = (1 / m_samples) * np.dot(X.T, (y_hat - y)) + reg_lambda * w / m_samples
    db = (1 / m_samples) * np.sum(y_hat - y)
    return dw, db
```

```
# Update params
```

```
def update_params(w, b, dw, db, lr):
    w -= lr * dw
    b -= lr * db
    return w, b
```

```
# Train function
```

```
def train(X, y, lr, n_epochs, X_val=None, y_val=None):
    m_samples, m_features = X.shape
    w, b = initialize_params(m_features, 35)
    train_costs = []
    val_costs = []

    for epoch in range(n_epochs):
        y_hat = forward(X, w, b)
        cost = compute_cost(y, y_hat, w)
        dw, db = compute_gradients(X, y, y_hat, w, reg_lambda=0.01)
        w, b = update_params(w, b, dw, db, lr)

        train_costs.append(cost)

        if X_val is not None and y_val is not None:
            y_val_hat = forward(X_val, w, b)
            val_cost = compute_cost(y_val, y_val_hat, w)
            val_costs.append(val_cost)

    return w, b, train_costs, val_costs
```

```
# Predict function
```

```
def predict_proba(X, w, b):
    return forward(X, w, b)
```

```
# Finish preparing data
X = pd.get_dummies(train_data.drop("Survived", axis=1)).values.astype(float)
y = train_data["Survived"].values.reshape(-1, 1).astype(float)
```

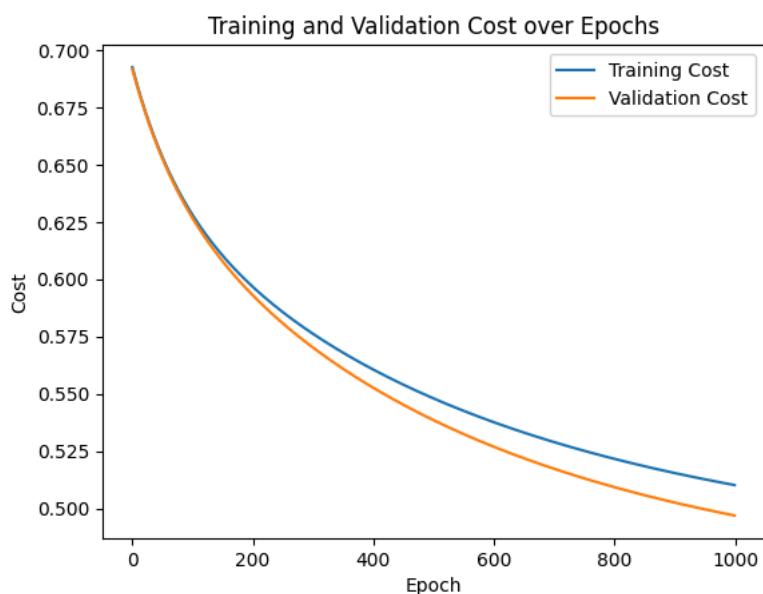
Double-click (or enter) to edit

```
#Split data into training and validation sets
X_train, X_val, y_train, y_val = sk.model_selection.train_test_split(X, y, test_size=0.2, random_state=42)
```

```
import matplotlib.pyplot as plt

# Train model and plot costs
w, b, train_costs, val_costs = train(X_train, y_train, lr=0.01, n_epochs=1000, X_val=X_val, y_val=y_val)

plt.plot(train_costs, label='Training Cost')
plt.plot(val_costs, label='Validation Cost')
plt.xlabel('Epoch')
plt.ylabel('Cost')
plt.title('Training and Validation Cost over Epochs')
plt.legend()
plt.show()
```



```
#drop the first column in X_test to match training data
X_test = test_df.drop("PassengerId", axis=1).values.astype(float)

# Make predictions on test set and save to CSV
y_test_pred_proba = predict_proba(X_test, w, b)
y_test_pred = (y_test_pred_proba >= 0.5).astype(int)
output = pd.DataFrame({'PassengerId': test_df.PassengerId, 'Survived': y_test_pred.flatten()})
output.to_csv('submission.csv', index=False)
```

