

HW2 Prompt History



Summary of Conversation:

This project focused on predicting passenger survival in the Titanic dataset by applying machine learning techniques, following the **CRISP-DM** framework. Each stage—from business understanding to deployment—was addressed by refining the model based on user prompts, exploring various machine learning methods, performing feature selection, optimizing model parameters, and generating submission files for Kaggle. Each stage was improved through iterative refinements via prompts, ensuring a complete end-to-end machine learning project ready for evaluation and submission.

History of Prompts and Responses

Below is the complete list of prompts, responses, and code generated during each stage of the CRISP-DM process.

Step 1: Business Understanding

Prompt_Step1:

vbnet

I'm asked to do a project using logistic regression to analyze Titanic dataset and submit my result on test dataset to the competition. Please review your setting and make a plan.

Response_Step1: Outlined the CRISP-DM framework and plan for the project, covering business understanding, data understanding, data preparation, modeling, evaluation, and deployment steps.

Step 2: Data Understanding

Prompt_Step2:

vbnet

Let's move to Step 2 for data understanding. Please also give an explanation on the features, for example, the meaning of each feature, its data type, etc. EDA is also needed to help me understand the data. Finally, please give initial insights on the data.

Please perform EDA on all features and gather the plots into a few canvases. For example, use one canvas to show all categorical data and one to show all continuous data for easier comparison.

Additional requirements:

- Make sure it covers all the features.
- Analyze by survival or not when performing the count by features.

Response_Step2:

- Provided detailed feature explanations, covering all features and data types.
- Generated EDA code to analyze all features, separating categorical and continuous variables into two canvases.

Code:

python

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load Titanic dataset
train_data = pd.read_csv('train.csv')

# Handle missing values for visualization (imputing age with median and embarking with mode)
train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
train_data['Embarked'].fillna(train_data['Embarked'].mode()[0], inplace=True)
train_data['Cabin'] = train_data['Cabin'].fillna('Unknown')

# Categorical features analysis
categorical_features = ['Pclass', 'Sex', 'Embarked', 'SibSp', 'Parch', 'Cabin', 'Ticket']
fig, axes = plt.subplots(4, 2, figsize=(16, 20))
fig.suptitle('Categorical Feature Analysis by Survival Status', fontsize=18)
for idx, feature in enumerate(categorical_features):
    ax = axes[idx // 2, idx % 2]
    sns.countplot(x=feature, hue='Survived', data=train_data, ax=ax)
    ax.set_title(f'Survival Count by {feature}')
fig.delaxes(axes[3, 1]) # Remove empty subplot
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()

# Continuous features analysis
continuous_features = ['Age', 'Fare']
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle('Continuous Feature Analysis by Survival Status', fontsize=18)
for idx, feature in enumerate(continuous_features):
    ax = axes[idx]
    sns.histplot(data=train_data, x=feature, hue='Survived', multiple='stack', kde=True, ax=ax)
    ax.set_title(f'Survival Distribution by {feature}')
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Step 3: Data Preparation

Prompt_Step3:

rust

OK, let's move to Step 3. Don't miss any features.

Response_Step3:

- Listed all original and engineered features.
- Prepared data for modeling by handling missing values, feature engineering, encoding, and scaling.

Code:

python

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer

train_data = pd.read_csv('train.csv')

# 1. Handle missing values
train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
train_data['Embarked'].fillna(train_data['Embarked'].mode()[0], inplace=True)
train_data['Cabin'] = train_data['Cabin'].fillna('Unknown')
train_data['Deck'] = train_data['Cabin'].str[0]

# 2. Feature Engineering
train_data['Title'] = train_data['Name'].str.extract('([A-Za-z])\.', expand=False)
train_data['FamilySize'] = train_data['SibSp'] + train_data['Parch'] + 1
train_data['IsAlone'] = (train_data['FamilySize'] == 1).astype(int)
train_data['FarePerPerson'] = train_data['Fare'] / train_data['FamilySize']

# 3. Encode categorical variables
train_data = pd.get_dummies(train_data, columns=['Pclass', 'Sex', 'Embarked', 'Title', 'Deck'], drop_first=True)
train_data.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# 4. Scale features
scaler = StandardScaler()
train_data[['Age', 'Fare', 'FarePerPerson']] = scaler.fit_transform(train_data[['Age', 'Fare', 'FarePerPerson']])

X_train = train_data.drop('Survived', axis=1)
y_train = train_data['Survived']
```

Step 4-1: Initial Modeling and Feature Selection

Prompt_Step4-1:

vbnet

Let's move to Step 4 and break it down step by step:
1. Build an initial model and validate the initial model performance.
2. Perform feature selection using the following methods to select n (could be set by user) most important features:
- RFE
- SelectKBest
- Optuna
Please consider that there's some restriction with SelectKBest, as the input x seems cannot be negative value.
Finally, please select features according to these three methods (union) for the next stage.

Response_Step4-1: Built the initial logistic regression model, validated it, and conducted feature selection using RFE, SelectKBest, and Optuna.

Code:

python

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE, SelectKBest, f_classif
from sklearn.preprocessing import MinMaxScaler
import optuna

# Initial Logistic Regression model validation
model = LogisticRegression(max_iter=1000, random_state=42)
initial_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
print(f"Initial Logistic Regression Accuracy: {initial_scores.mean():.4f}")

# 1. RFE
rfe = RFE(estimator=model, n_features_to_select=10)
rfe.fit(X_train, y_train)
rfe_features = X_train.columns[rfe.support_].tolist()

# 2. SelectKBest with MinMax scaling
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
skb = SelectKBest(score_func=f_classif, k=10)
skb.fit(X_train_scaled, y_train)
skb_features = X_train_scaled.columns[skb.get_support()].tolist()

# 3. Optuna-based selection
def objective(trial):
    selected_features = [i for i in range(X_train.shape[1]) if trial.suggest_categorical(f'feature_{i}', [0, 1]) == 1]
    if not selected_features: return 1.0
    model = LogisticRegression(max_iter=1000, random_state=42)
    X_selected = X_train.iloc[:, selected_features]
    scores = cross_val_score(model, X_selected, y_train, cv=5, scoring='accuracy')
    return -scores.mean()

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)
optuna_features = [X_train.columns[i] for i in range(X_train.shape[1]) if study.best_trial.params[f'feature_{i}'] == 1]

# Union of selected features
selected_features = list(set(rfe_features) | set(skb_features) | set(optuna_features))
X_train_selected = X_train[selected_features]
```

Step 4-2: Model Selection with Random Forest

Prompt_Step4-2:

vbnet

For the next stage of step 4, suggest another model other than logistic regression using selected features. I want to get the best accuracy on the test set.

Response_Step4-2: Recommended using **Random Forest** with hyperparameter tuning for better accuracy on the test set.

Code:

python

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Hyperparameter tuning for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42), param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_selected, y_train)

# Train best model
best_rf = grid_search.best_estimator_
X_test_selected = X_test[selected_features]
y_pred = best_rf.predict(X_test_selected)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Set Accuracy: {test_accuracy:.4f}")
```

Step 5: Evaluation

Prompt_Step5:

vbnet

Please explain the full process of Step 5, including evaluating the model, generating a confusion matrix, additional metrics, and fine-tuning the model. Also, compare the approach with champions' solutions from Kaggle and point out improvements based on their code. Share the links to the champion solutions you referred to.

Response_Step5: Evaluated model performance using multiple metrics and proposed improvements by analyzing champion solutions on Kaggle.

Code for Evaluation:

python

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate accuracy
test_accuracy = accuracy_score(y_test, y_pred)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()

# Additional metrics
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, best_rf.predict_proba(X_test_selected)[:, 1])
print(f"Precision: {precision:.4f}, Recall: {recall:.4f}, F1 Score: {f1:.4f}, ROC-AUC: {roc_auc:.4f}")
```

Step 6: Deployment

Prompt_Step6:

vbnet

Finally, let's move to Step 6. Show me how to output my prediction result and submit it on Kaggle.

Response_Step6: Generated submission file for Kaggle.

Code for Submission:

python

```
submission = pd.DataFrame({
    'PassengerId': test_data['PassengerId'],
    'Survived': y_pred
})
submission.to_csv('titanic_submission.csv', index=False)
```

This history showcases a step-by-step implementation of the Titanic project using CRISP-DM, demonstrating iterative development and model refinement through prompt-based interaction.