3.1.3.3. Documentação do Processo de construção e Treinamento do Modelo

Documentação do Processo de Construção e Treinamento do Modelo
Introdução:
Este documento fornece uma visão detalhada do processo de construção e treinamento do modelo de aprendizado de máquina para a tarefa específica. Descreve as etapas, parâmetros selecionados e os resultados obtidos durante o desenvolvimento do modelo.
Objetivo:
O objetivo principal deste modelo é [descrever brevemente o objetivo do modelo, por exemplo, prever vendas futuras, classificar dados, etc.].

Etapas do Processo:
1. Exploração de Dados e Pré-processamento:
Coleta de Dados:
Descreva as fontes de dados utilizadas.
Lista de variáveis/features incluídas.
Limpeza e Pré-processamento:
Identificação e tratamento de valores ausentes, outliers, etc.
Transformações aplicadas aos dados.
2. Implementação de Modelos de Aprendizado de Máquina:
Escolha de Algoritmos:
Justificativa para a escolha dos algoritmos utilizados.
Implementação:
Detalhes sobre como os modelos foram implementados.
Utilização de bibliotecas (por exemplo, Scikit-learn, TensorFlow).
3. Otimização e Validação do Modelo:
Otimização de Hiperparâmetros:
Descrição do processo de otimização.
Lista dos hiperparâmetros ajustados.
Validação Cruzada:
Detalhes sobre como a validação cruzada foi realizada.
Resultados obtidos.
Parâmetros do Modelo:
Lista completa de hiperparâmetros e seus valores finais após a otimização.
Outros parâmetros relevantes para o modelo.
Métricas de Avaliação:
Descrição das métricas utilizadas para avaliar o desempenho do modelo.
Resultados específicos obtidos para cada métrica.

3.2. Ciência de Dados

3.2.1. Análise Descritiva dos Dados

3.2.1.1. Utilização de Técnicas Estatísticas Básicas para descrever os Dados

```
import pandas as pd
import numpy as np

# Carregue o conjunto de dados
dados = pd.read_csv('caminho/para/seu/arquivo/car_rental_data.csv')

# Supondo que queremos calcular as estatísticas de uma coluna específica, por exemplo, 'demand'
coluna_dados = dados['demand']

# Calculando a média
media = np.mean(coluna_dados)
print(f"Média: {media}")

# Calculando a mediana
mediana = np.median(coluna_dados)
print(f"Mediana: {mediana}")

# Calculando o desvio padrão
desvio_padrao = np.std(coluna_dados)
print(f"Desvio Padrão: {desvio_padrao}")
```

3.2.1.2. Visualização de Dados utilizando Bibliotecas como Matplotlib e Seaborn

```
import matplotlib.pyplot as plt

# Supondo que 'date' seja a coluna de datas e 'demand' a coluna de demanda
plt.figure(figsize=(10, 6))
plt.plot(dados['date'], dados['demand'])
plt.xlabel('Data')
plt.ylabel('Demanda')
plt.title('Demanda ao longo do tempo')
plt.xticks(rotation=45)
plt.show()
```

Gráfico de dispersão (Scatter plot):

```
import seaborn as sns
```

```
# Supondo que 'temperature' seja uma coluna de temperatura que influencia a demanda
plt.figure(figsize=(10, 6))
sns.scatterplot(x='temperature', y='demand', data=dados)
plt.xlabel('Temperatura')
plt.ylabel('Demanda')
plt.title('Demanda vs Temperatura')
plt.show()
```

3.2.1.3. Identificação de Padrões e tendências nos Dados

Identificações de Outliers:

```
plt.figure(figsize=(10, 6))
sns.boxplot(y='demand', data=dados)
plt.title('Identificação de Outliers na Demanda')
plt.show()
```

Analise de padrões sazonais:

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='day_of_week', y='demand', data=dados)
plt.xlabel('Dia da Semana')
plt.ylabel('Demanda')
plt.title('Demanda por Dia da Semana')
plt.show()
```

3.2.2. Modelagem Estatística

3.2.2.1. Aplicação de técnicas estatísticas avançadas para modelagem dos dados

```
import pandas as pd

import statsmodels.api as sm

from scipy import stats


# Carregar o conjunto de dados

dados = pd.read_csv('caminho/para/seu/arquivo/car_rental_data.csv')
```

```
# Regressão Linear

X = dados['temperature']

y = dados['demand']

X = sm.add_constant(X)

modelo = sm.OLS(y, X).fit()

print(modelo.summary())



grupos = [dados[dados['month'] == mes]['demand'] for mes in
dados['month'].unique()]

anova_resultado = stats.f_oneway(*grupos)

print('ANOVA F-statistic:', anova_resultado.statistic)

print('ANOVA p-value:', anova_resultado.pvalue)
```

3.2.2.2. Uso de ferramentas como regressão linear, classificação etc.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Carregar o conjunto de dados
dados = pd.read_csv('caminho/para/seu/arquivo/car_rental_data.csv')

# Regressão Linear
X_lr = dados[['temperature']]

y_lr = dados['demand']


X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X_lr, y_lr,
test_size=0.2, random_state=42)
```

```
modelo_lr = LinearRegression()

modelo_lr.fit(X_train_lr, y_train_lr)

y_pred_lr = modelo_lr.predict(X_test_lr)

mse_lr = mean_squared_error(y_test_lr, y_pred_lr)

r2_lr = r2_score(y_test_lr, y_pred_lr)

print(f'Regressão Linear - MSE: {mse_lr}, R-squared: {r2_lr}')

# Classificação
dados['demand_category'] = pd.cut(dados['demand'], bins=[0, 100, 200, 300],
labels=['Baixa', 'Média', 'Alta'])

X_clf = dados[['temperature', 'humidity', 'windspeed']]

y_clf = dados['demand_category']

X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X_clf, y_clf,
test_size=0.2, random_state=42)

scaler = StandardScaler()

X_train_clf = scaler.fit_transform(X_train_clf)

X_test_clf = scaler.transform(X_test_clf)

modelo_rf = RandomForestClassifier(random_state=42)

modelo_rf.fit(X_train_clf, y_train_clf)

y_pred_clf = modelo_rf.predict(X_test_clf)

print(confusion_matrix(y_test_clf, y_pred_clf))

print(classification_report(y_test_clf, y_pred_clf))
```

3.2.2.3. Avaliação da adequação dos modelos estatísticos aos dados

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix,
classification_report, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

```
# Carregar o conjunto de dados
dados = pd.read_csv('caminho/para/seu/arquivo/car_rental_data.csv')

# Regressão Linear
X_lr = dados[['temperature']]
y_lr = dados['demand']

X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X_lr, y_lr,
test_size=0.2, random_state=42)

modelo_lr = LinearRegression()

modelo_lr.fit(X_train_lr, y_train_lr)

y_pred_lr = modelo_lr.predict(X_test_lr)

mse_lr = mean_squared_error(y_test_lr, y_pred_lr)
r2_lr = r2_score(y_test_lr, y_pred_lr)

print(f'Regressão Linear - MSE: {mse_lr}, R-squared: {r2_lr}')

# Classificação
dados['demand_category'] = pd.cut(dados['demand'], bins=[0, 100, 200, 300],
labels=['Baixa', 'Média', 'Alta'])

X_clf = dados[['temperature', 'humidity', 'windspeed']]

y_clf = dados['demand_category']

X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X_clf, y_clf,
test_size=0.2, random_state=42)

scaler = StandardScaler()

X_train_clf = scaler.fit_transform(X_train_clf)

X_test_clf = scaler.transform(X_test_clf)

modelo_rf = RandomForestClassifier(random_state=42)
modelo_rf.fit(X_train_clf, y_train_clf)

y_pred_clf = modelo_rf.predict(X_test_clf)

print(confusion_matrix(y_test_clf, y_pred_clf))

print(classification_report(y_test_clf, y_pred_clf))

if len(y_test_clf.unique()) == 2:
    y_prob_clf = modelo_rf.predict_proba(X_test_clf)[:, 1]
```

```
auc_roc = roc_auc_score(y_test_clf, y_prob_clf)

print(f'AUC-ROC: {auc_roc}')

fpr, tpr, _ = roc_curve(y_test_clf, y_prob_clf)
plt.figure(figsize=(10, 6))

plt.plot(fpr, tpr, marker='.')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Curva ROC')

plt.show()
```

3.2.2.4. Implementação de modelos preditivos utilizando Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix,
classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Carregar os dados
dados = pd.read_csv('caminho/para/seu/arquivo/car_rental_data.csv')

# Verificar os primeiros registros
print(dados.head())

# Preparação básica (ajuste conforme necessário)
dados = dados.dropna()

# Análise de correlação
sns.pairplot(dados[['temperature', 'humidity', 'windspeed', 'demand']])
plt.show()

# Calcular a correlação
correlation_matrix = dados.corr()
print(correlation_matrix)

# Regressão Linear
X = dados[['temperature', 'humidity', 'windspeed']]
y = dados['demand']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
modelo_lr = LinearRegression()
modelo_lr.fit(X_train, y_train)
y_pred_lr = modelo_lr.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
print(f'MSE (Linear Regression): {mse_lr}')
print(f'R² (Linear Regression): {r2_lr}')

# Classificação
dados['demand_category'] = pd.cut(dados['demand'], bins=[0, 100, 200, 300],
labels=['Baixa', 'Média', 'Alta'])
y_clf = dados['demand_category']
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X, y_clf, test_size=0.2,
random_state=42)
modelo_rf = RandomForestClassifier(random_state=42)
modelo_rf.fit(X_train_clf, y_train_clf)
y_pred_clf = modelo_rf.predict(X_test_clf)
print(confusion_matrix(y_test_clf, y_pred_clf))
print(classification_report(y_test_clf, y_pred_clf))
```

3.2.2.5. Avaliação da performance dos modelos preditivos

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix,
classification_report, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Carregar os dados
dados = pd.read_csv('caminho/para/seu/arquivo/car_rental_data.csv')

# Preparação básica
dados = dados.dropna()

# Dividir os dados em conjuntos de treino e teste
X = dados[['temperature', 'humidity', 'windspeed']]
y = dados['demand']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Regressão Linear
modelo_lr = LinearRegression()
modelo_lr.fit(X_train, y_train)
y_pred_lr = modelo_lr.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
```

```
r2_lr = r2_score(y_test, y_pred_lr)
print(f'Mean Squared Error (MSE): {mse_lr}')
print(f'R-squared (R²): {r2_lr}')

# Classificação
dados['demand_category'] = pd.cut(dados['demand'], bins=[0, 100, 200, 300],
labels=['Baixa', 'Média', 'Alta'])
y_clf = dados['demand_category']
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(X, y_clf, test_size=0.2,
random_state=42)
modelo_rf = RandomForestClassifier(random_state=42)
modelo_rf.fit(X_train_clf, y_train_clf)
y_pred_clf = modelo_rf.predict(X_test_clf)
print(confusion_matrix(y_test_clf, y_pred_clf))
print(classification_report(y_test_clf, y_pred_clf))

# AUC-ROC (para problemas binários)
if len(y_test_clf.unique()) == 2:
    y_prob_clf = modelo_rf.predict_proba(X_test_clf)[:, 1]
    auc_roc = roc_auc_score(y_test_clf, y_prob_clf)
    print(f'AUC-ROC: {auc_roc}')
    fpr, tpr, _ = roc_curve(y_test_clf, y_prob_clf)
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, marker='.')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Curva ROC')
    plt.show()
```

3.2.2.6. Comparação entre diferentes abordagens de análise preditiva

Redes Neurais X Forest

```
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier

# Exemplo de Rede Neural
mlp_model = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)
mlp_model.fit(X_train, y_train)
mlp_predictions = mlp_model.predict(X_test)
mlp_accuracy = accuracy_score(y_test, mlp_predictions)

# Exemplo de Random Forest
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
```

```
print("Rede Neural Accuracy:", mlp_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
```

Métodos Ensemble X Modelos Individuais

```
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score

# Exemplo de Modelo Ensemble (Votação)
voting_model = VotingClassifier(estimators=[('lr', lr_model), ('dt', dt_model), ('svm',
svm_model)], voting='hard')
ensemble_scores = cross_val_score(voting_model, X_train, y_train, cv=5,
scoring='accuracy')

# Exemplo de Modelo Individual
individual_scores = cross_val_score(svm_model, X_train, y_train, cv=5,
scoring='accuracy')

print("Ensemble Mean Accuracy:", ensemble_scores.mean())
print("Individual SVM Mean Accuracy:", individual_scores.mean())
```