Data Mining 4th year Software Engineering

Duration: 3 hours
2025/2026

TP3 : Data Preprocessing

Dataset: insurance_claims.csv

Tools: Python, Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn

1. Data Preprocessing:

This is the first step in building a machine learning model. Data pre-processing refers to the transformation of data, before feeding it into the model. It deals with the techniques that are used to convert unusable raw data into clean reliable data.

Task 1 import the basic libraries

```
 # Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set()  #  Will import Seaborn functionalities
# we don't like warnings
# you can comment the following 2 lines if you'd like to
import warnings
warnings.filterwarnings('ignore')
```

Task 2 Load the Data

1. Read the CSV file into a DataFrame named autinsurance:

```
autinsurance = pd.read_csv('insurance_claimsV3.csv')
autinsurance.head()
```

2. Display:
   - The first 5 rows (head()).
   - The shape of the dataset (.shape).

- The column names (.columns).

Task 3 Distribution of target variable

1. Use autinsurance.info() to inspect data types and non-null counts.
2. Use autinsurance.describe() on numerical columns.
3. Compute the distribution of the target variable:

```
autinsurance['fraud_reported'].value_counts()
```

4. To calculate fractions, pass `normalize=True` to the `value_counts` function.

```
autinsurance['fraud_reported'].value_counts(normalize=True)
```

5. Plot the distribution of the target variable as a bar plot.

```
(autinsurance['fraud_reported'].value_counts().plot(
      kind='bar',
      figsize=(8, 6),
      title='Distribution of Target Variable',
  )
 );
 plt.show()
```

Task 4. Handling Missing Values

4.1 Detect Missing Values

1. Start by checking missing values:

```
# Disply all Columns
pd.options.display.max_rows=170

# Finding missing values

autinsurance.isna().sum()
```

2. Delete rows with NaN values Once you have figured out all the missing details, we remove all the missing rows from the DataFrame. we use the dropna() function:

```
# removing Null values
autinsurance = autinsurance.dropna()
autinsurance.info()
```

3. Which columns contain missing values?

4.2 Convert Special Markers to NaN

1. Replace the ? values by np.nan in the appropriate columns, for example:

```
autinsurance["property_damage"] =
autinsurance["property_damage"].replace("?", np.nan)
autinsurance["police_report_available"] =
autinsurance["police_report_available"].replace("?", np.nan)
```

2. Recompute autinsurance.isna().sum(). How did the number of missing values change after replacing ? by NaN?

4.3 Delete Rows or Columns

1. Choose one strategy to handle missing values:
   - Drop rows with any missing values (e.g. dropna(axis=0, how="any")), OR
   - Drop columns with too many missing values.
2. Apply your chosen strategy and record how many rows/columns remain.
3. Do you think dropping rows is appropriate for this dataset? Why or why not?

4.4 Impute Missing Numerical Values

1. Impute the numerical data of the age column with its mean. To do so, first find the mean of the column with missing values using the mean() function of pandas, and then print it

```
col = "some_numeric_column"

mean_value = autinsurance[col].mean()

autinsurance[col].fillna(mean_value, inplace=True)
```

4.5 – Impute Missing Categorical Values with Mode

Example: `insured_sex`

1. Compute the mode of `insured_sex`:

```
mode_gender = autinsurance["insured_sex"].mode()[0]
```

2. Fill missing values in `insured_sex` with this mode:

```
autinsurance["insured_sex"].fillna(mode_gender,
inplace=True)
```

Task 5 Outlier Detection and Treatment

5.1 Visualize Outliers with Boxplots

1. Select 2–3 numerical features (e.g. total_claim_amount, injury_claim, etc.).
2. Plot boxplots using either pandas or seaborn:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x=autinsurance["total_claim_amount"])
plt.show()
```

```
# Boxplots
# Horizontal boxplot with observations
plt.rcParams['figure.figsize'] = 8,4
sns.boxplot(autinsurance['months_as_customer'])
```

3. Which variables contain clear outliers?
4. How do these outliers appear in the boxplot?

5.2 IQR-Based Outlier Detection

- Examine the distribution of the data in column 4

Quartile 1: from 18 to 32

Quartile 2: from 32 to 37

Quartile 3: from 37 to 44

Quartile 4: from 44 to 92

Quartile 4 is much larger than the other quartiles. This raises the possibility of outliers.

A Quantile - Quantile (qq) plot can help identify outliers

```
import numpy as np
import pylab
import scipy.stats as stats
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
%matplotlib inline

stats.probplot(autinsurance['policy_annual_premium'], dist="norm",
plot=pylab)
pylab.show()
```

1. How many outliers are detected using the IQR rule?

```
Q1 = autinsurance["months_as_customer"].quantile(0.25)

Q3 = autinsurance["months_as_customer"].quantile(0.75)

IQR = Q3 - Q1

print(IQR)
```

```
# Test if outliers are fixed

plt.rcParams['figure.figsize'] = 4,8
sns.boxplot(x='months_as_customer', data=au_copy, orient = 'v');
```

5.3 Z-Score Based Outlier Detection

1. The intuition behind Z-score is to describe any data point by finding their relationship with the Standard Deviation and Mean of the group of data points. Z-score is finding the distribution of data where mean is 0 and standard deviation is 1 i.e. normal distribution.

```
from scipy import stats
import numpy as np

autinsurance_num = autinsurance.select_dtypes(include=[np.number])

z = np.abs(stats.zscore(autinsurance_num))
print(z)
```

```
autinsurance_num.head()
```

2. While calculating the Z-score we re-scale and center the data and look for data points which are too far from zero. These data points which are way too far from zero will be treated as the outliers. In most of the cases a threshold of 3 or -3 is used i.e if the Z-score value is greater than or less than 3 or -3 respectively, that data point will be identified as outliers.

```
outliers = np.where(z > 3)
outliers
```

note: Don't be confused by the results. The first array contains the list of row numbers and second array respective column numbers, which mean z[31][1] have a Z-score higher than 3.

3. Are the outliers detected by z-score similar to those detected by the IQR method?

Task 6. Feature Selection

6.1 Correlation-Based Feature Selection

Collinear Features : are features that are highly correlated with one another. In machine learning, these lead to decreased generalization performance on the test set due to high variance and less model interpretability.

The identify_collinear method finds collinear features based on a specified correlation coefficient value. For each pair of correlated features, it identifies one of the features for removal (since we only need to remove one):

1. Create a correlation matrix for numerical features:

```
# Create correlation matrix
au_copy = autinsurance.copy()
corr_matrix = au_copy.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.7
to_drop = [column for column in upper.columns if any(upper[column] >
0.7)]
```

2. Which features are strongly correlated with each other?
3. Remove features that have a correlation > 0.7 with another feature (highly correlated pairs).
4. Which features did you decide to drop due to multicollinearity?

```
# Correlation with Target Variable

corr_matrix = au_copy.corr().abs()


plt.figure(figsize=(30,20))
au_copy.corr()['fraud_reported'].plot()
#autinsurance.corr()[:].plot()
#autinsurance.corr().plot()
```

6.2 Model-Based Feature Selection

1. Encode categorical variables (e.g. using pd.get_dummies) to create a numeric dataset.

```
autinsurance_v3 = au_copy.copy()
```

```
autinsurance_v3.shape
```

```
print(autinsurance_v3.columns)
```

2. Split the dataset into features X and target y:

```
X = autinsurance_v3.drop("fraud_reported", axis=1)
y = autinsurance_v3["fraud_reported"]
```

```
X.shape
y.shape
```

3. Splitting the dataset into the Training set and Test set

```
  # Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split

training_features, test_features, \
training_target, test_target, =
train_test_split(autinsurance_v3.drop('fraud_reported', axis = 1),
                                                    autinsurance_v3.fraud_re
ported,
                                                    test_size = .2,
                                                    random_state=12)
```

4. Train a simple model such as XGboost inspect **feature importances** or coefficients.

```
 from xgboost import XGBClassifier

from matplotlib import pyplot

plt.figure(figsize=(20,10))

# fit model no training data
model = XGBClassifier()
model.fit(training_features, training_target)

# feature importance
#print(model.feature_importances_)

# plot
```

```
pyplot.bar(range(len(model.feature_importances_)),
model.feature_importances_)
pyplot.show()
```

5. Which features are most important according to the model?

```
feature_importances = pd.DataFrame({'Importance Coef'
:model.feature_importances_ , 'Features' : training_features.columns})
feature_importances.nlargest(135, 'Importance Coef')
```

7. Save the Prepared Dataset

1. After all preprocessing steps (missing values, outliers, feature selection), create a final version of the dataset, e.g.:

```
autinsurance_clean = autinsurance.copy()
```

2. Save it to disk:

```
# Save to file
autinsurance_v3.to_csv("insurance_claimsV4.csv")
```

3. What is the final number of features (columns) and rows?

```
autinsurance_v3.shape
```