Data Mining 4th year
Software Engineering

Duration: 3 hours
2025/2026

TP3_Session 2: Data transformation

Dataset: insurance_claimsV3.csv

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

1. Importing the main 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')
```

2. Read the data

```
autinsurance = pd.read_csv('insurance_claimsV3.csv')#.drop('Unnamed:
0', axis = 1)
autinsurance.head()
```

3. Data Proportion

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

4. Identify features and target variables

```
X = autinsurance.drop('fraud_reported', axis = 1) # axis = 1 (look in
columns) OR axis = 0  (look in rows)

y = autinsurance.fraud_reported
```

5. Data split & Scaling Data Preprocessing

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
#from sklearn.cross_validation import train_test_split
training_features, test_features, \
training_target, test_target, = train_test_split(X,y, test_size = .2,
random_state = 42)
```

6. Using a Dummy Classifier

6.1 As a first classifier, you can apply the built-in [`DummyClassifier` class from `sklearn.dummy`](https://scikitlearn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html) to set a baseline for performance of our future models. This classifier does not actually use the feature matrix `X_digits_train`; classification decisions are made using the target vector `y_digits_train` only. There are a few strategies, but we'll start with the `**'most_frequent'**` strategy. That is, the `predict` method always returns the majority class.

```
from sklearn.dummy import DummyClassifier

dummy_baseline = DummyClassifier(strategy="most_frequent")# all 0

dummy_baseline.fit(training_features, training_target)
```

6.2 Having applied the `fit` method to the training data, you can use the `predict` method to see how this estimator classifies the data. Unsurprisingly, it returns a vector of all `0 s (because that is the majority class for this data).

```
test_target_pred = dummy_baseline.predict(test_features)
print(test_target_pred)
```

6.3 You can find the fraction of correct classifications using the method `score` with the test data:

```
score = dummy_baseline.score(test_features, test_target)
print('The fraction of correct classifications is:
{:5.3f}'.format(score))
```

Using `dummy.score` is equivalent to explicitly comparing the entries of `y_digits_pred` to `y_digits_test`, counting the number of correct classifications, and dividing by the number of classifications in total.

7. confusion_matrix

In Scikit-Learn, the confusion_matrix function takes as arguments the actual labels followed by the predicted labels (labelled in ascending order according to the class labels). From the documentation:

sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None, sample_weight=None)

Compute confusion matrix to evaluate the accuracy of a classification

By definition a confusion matrix C is such that Ci,j is equal to the number of observations known to be in group i but predicted to be in group j.

Thus in binary classification, the count of true negatives is C0,0, false negatives is C1,0, true positives is C1,1, and false positives is C0,1.

```
# This is the long way of computing the accuracy score
from sklearn.metrics import  confusion_matrix
dummy_baselineCM = confusion_matrix(test_target,test_target_pred)
dummy_baselineCM
```

Try this :

```
# from sklearn.linear_model import Perceptron
# classifier = Perceptron()
# classifier.fit(training_features, training_target)
# accuracy = classifier.score(test_features, test_target)
# print("Prediction Accuracy:{:.2f}%".format(accuracy * 100))
```

8. Building Model (Decision Tree)

8.1 First Method (using function)

```

```

```
from sklearn.tree import DecisionTreeClassifier as Model
```

```
def train(features, target):
     model = Model()
    model.fit(features, target)
   return model
```

```
def predict(model, new_features):
  preds = model.predict(test_features)
  return preds
```

```
# Assume Titanic data is loaded into titanic_feats,
 # titanic_target and titanic_test
 model = train(training_features, training_target)
predictions = predict(model, test_features)
```

8.2 Second Method

```
from sklearn.tree import DecisionTreeClassifier as dtc
```

```
DecisionTreeModel = dtc(criterion='entropy', max_depth = 2,
random_state=42)#

DecisionTreeModel
```

```
%%time
DecisionTreeModel.fit(training_features, training_target)   # Training
input and its Target variables
```

```
DT_Pred = DecisionTreeModel.predict(test_features) # I already Know
y_test  # 200 variables

DT_Pred  # Prediction for the 20%
```

```
# Need to compate the hidden target varialbe (Test_target) with what
the model is going to get out of prediction

test_target  # y of the 20%
```

Question1: in a short paragraph explain the difference between the first and the second method

8.3 Making the Confusion Matrix

```
# Confusion Matrix
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

```
# Import machine learning modules
from sklearn.ensemble import GradientBoostingClassifier,
partial_dependence
```

```
# Confusion Matrix
CMTD = confusion_matrix(test_target,DT_Pred) # Compare the predicted
target varaible to the orginal target variable
CMTD
```

```
# #target = 'fraud_reported'
# CMTD = pd.crosstab(test_target,DT_Pred, rownames=['Actual'],
colnames=['Predicted'])
# fig, (ax1) = plt.subplots(ncols=1, figsize=(5,5))
# sns.heatmap(CMTD,
#             xticklabels=['Fraudulant', 'Legit'],
#             yticklabels=['Fraudulant', 'Legit'],
#             annot=True,ax=ax1,
#             linewidths=.2,linecolor="Darkblue", cmap="Blues")
# plt.title('Confusion Matrix', fontsize=14)
# plt.show()
```

Question 2 : explain how to make a confusion matrix.

```
# Accuracy Score
ADT= accuracy_score(test_target, DT_Pred)
```

```
print(" Decision Tree Prediction Accuracy : {:.2f}%".format(ADT * 100))
# print()
```

9. **Model Evaluation:** Computing the Accuracy of a Binary Classifier

The most basic way to assess performance is to compare the total number of correct predictions and the total number of observations. This is the **accuracy**. Using the diagram of our confusion matrix above, the accuracy can be written as

$$\text{accuracy} = \frac{\mathbf{tp} + \mathbf{tn}}{\mathbf{tn} + \mathbf{tp} + \mathbf{fn} + \mathbf{fp}}$$

where **tn , tp, fn** and **fp** are the number of true negatives, true positives, false negatives, and false positives respectively.

```
# from sklearn.metrics import classification_report
# #d = DecisionTreeModel.fit(X_train.values, y_train.values.copy(), 50)
# #
# #preds = predict(X_test, d)
# print(classification_report(test_target,DT_Pred))
```

Question 3: cite another evaluation metrics

9.2 feature-importance

A fantastic characteristic of many ensemble models is that you can interpret the feature importance. As you learned with Decision Trees, the most important

features are selected first during the construction of a tree. Using the gini or information gain generated from using a feature to make a split, a feature importance score can be calculated.

In the case of ensembles, these feature importance scores are aggregated over all of the trees within the ensemble. `scikit-learn` conveniently calculates a `.feature_importance_` score for many of their ensemble implementations.

```
#DecisionTreeModel.fit(training_features, training_target)
### Verification:
results = pd.DataFrame(index= training_features.columns,
data={'importance':DecisionTreeModel.feature_importances_})
print('Feature importances:\n{}'.format(results))
```

```
# Plot feature importances
plt.rcParams['figure.figsize'] = 20,30
plt.title('Normalized Feature Importances')
sns.barplot(x = DecisionTreeModel.feature_importances_, y
=training_features.columns, orient = 'h')
plt.show()
```

```
feature_importances = pd.DataFrame({'Importance Coef'
:DecisionTreeModel.feature_importances_ , 'Features' :
training_features.columns})
feature_importances.nlargest(10, 'Importance Coef')
```

```
features = feature_importances.nlargest(135, 'Importance Coef')

features =[x for x in features['Features'] if x!=0]
features
```

```
autinsurance = autinsurance[['insured_hobbies_chess',
 'vehicle_claim',
 'incident_severity_Total Loss',
 'insured_hobbies_cross-fit',
 'incident_severity_Minor Damage',
```

```
 'insured_hobbies_camping',
 'auto_model_Civic',
 'incident_state_WV',
 'insured_occupation_handlers-cleaners','fraud_reported']]
```

Question4: Explain the role of features importance step .

10. Plot Decision Tree

```
import os

os.environ['PATH'] =
os.environ['PATH']+';'+os.environ['CONDA_PREFIX']+r"\Library\bin\graphv
iz"
```

```
from six import StringIO
```

```
import six
import sys
sys.modules['sklearn.externals.six'] = six
```

```
#! pip install pydotplus
```

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
```

```
dot_data = StringIO()
export_graphviz(DecisionTreeModel, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =
X.columns,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Auto-Inssurance.png')
Image(graph.create_png())
```

Save the figure in your computer.