Machine Learning: Data Representation

CSC 640: Advanced Software Engineering

James Walden

Northern Kentucky University
Topics

1. Data Representation
2. Missing Data
3. Scaling
4. References
## Detecting Missing Data

```python
In [2]: df = pd.read_csv('titanic.csv')
In [3]: df.shape
Out[3]: (891, 12)
In [4]: df.isnull().sum()
Out[4]:
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PassengerId</td>
<td>0</td>
</tr>
<tr>
<td>Survived</td>
<td>0</td>
</tr>
<tr>
<td>Pclass</td>
<td>0</td>
</tr>
<tr>
<td>Name</td>
<td>0</td>
</tr>
<tr>
<td>Sex</td>
<td>0</td>
</tr>
<tr>
<td>Age</td>
<td>177</td>
</tr>
<tr>
<td>SibSp</td>
<td>0</td>
</tr>
<tr>
<td>Parch</td>
<td>0</td>
</tr>
<tr>
<td>Ticket</td>
<td>0</td>
</tr>
<tr>
<td>Fare</td>
<td>0</td>
</tr>
<tr>
<td>Cabin</td>
<td>687</td>
</tr>
<tr>
<td>Embarked</td>
<td>2</td>
</tr>
</tbody>
</table>

dtype: int64
```
1. Fix data collection problem that causes missing data.

2. Eliminate rows with missing values (dropna()).

3. Eliminate columns with too many missing values.


5. Numeric features: replace with mean, median, etc. value.
Drop all rows with missing data in one column

```python
In [10]: df.dropna(subset=['Embarked'], inplace=True)
In [11]: df.shape
Out[11]: (889, 12)
In [12]: df.isnull().sum()
Out[12]:
PassengerId  0
Survived     0
Pclass       0
Name         0
Sex          0
Age          177
SibSp        0
Parch        0
Ticket       0
Fare         0
Cabin        687
Embarked     0
dtype: int64
```
Drop column with many missing values

In [16]: df.drop("Cabin", axis=1, inplace=True)
In [17]: df.shape
Out[17]: (889, 11)
In [18]: df.isnull().sum()
Out[18]:
PassengerId   0
Survived      0
Pclass        0
Name          0
Sex           0
Age           177
SibSp         0
Parch         0
Ticket        0
Fare          0
Embarked      0
dtype: int64
Impute a value for missing data

In [20]: median = df["Age"].median()
In [21]: df["Age"].fillna(median, inplace=True)
In [22]: df.isnull().sum()
Out[22]:
PassengerId 0
Survived 0
Pclass 0
Name 0
Sex 0
Age 0
SibSp 0
Parch 0
Ticket 0
Fare 0
Embarked 0
dtype: int64
Impute a value for missing data

**Before Imputing Age**

**After Imputing Age**
Measurement Scales

- **Nominal Scale**: categorical labels with no notion of ordering among the categories. Example: classify faults as arising from requirements, design, or code.

- **Ordinal Scale**: categorical labels with ordering. Example: high, medium, low priority for defects.

- **Interval Scale**: preserves ordering and differences between items, so addition and subtraction are possible. Example: temperature.

- **Ratio Scale**: preserves ordering, differences, and ratios. Measurement must start at zero and increase at equal intervals. Example: length.

- **Absolute Scale**: measurements made by counting elements. All arithmetic operations possible. Example: LOC.
Data Representation

- **Absolute Scale**: float or integer.
- **Ratio Scale**: float or integer.
- **Interval Scale**: float or integer.
- **Ordinal Scale**: integer (high=3, medium=2, low=1).
- **Nominal Scale**: ?

While even ordinal data can be represented as a single numerical variable, nominal data has no ordering and needs a different representation, such as one-hot-encoding.
Detecting Categorical Features

In [42]: df["paymentMethod"].head()
Out[42]:
0      paypal
1  storecredit
2    creditcard
3    creditcard
4    creditcard
Name: paymentMethod, dtype: object
In [37]: df["paymentMethod"].value_counts()
Out[37]:
creditcard    28004
     paypal     9303
  storecredit    1914
Name: paymentMethod, dtype: int64
In [41]: df["paymentMethod"].value_counts().sum() == df.shape[0]
Out[41]: True
Nominal Feature Examples

**MachineType**
- Laptop
- Desktop
- Server

**CodeSignature**
- true
- false

**TopLevelDomain**
- .com
- .org
- .edu
- .gov
- .mil
- .info
- .io
- .ru
- .uk
One Hot Encoding

Replace categorical variable with one or more new binary-valued features. We transform the **defect-found-phase** categorical variable with 3 binary features, one for each category.

<table>
<thead>
<tr>
<th>requirements</th>
<th>design</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirements</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>design</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>code</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In Pandas, the `get_dummies(df, columns=['colname'])` function encodes categorical or string type columns using one-hot-encoding.
Many machine learning algorithms will give more importance to features with a larger range of values. The problem occurs when we have two features like:

- Size of file in bytes (0-1,000,000s)
- Number of dangerous library calls (0-100)

where the size scale is much larger than the scale for library calls.

We scale the features so that each feature contributes approximately proportionately to the prediction of the model.
Scaling Techniques

- **MinMax Scaling** shifts the data such that all features range between 0 and 1.

- **Standardization** ensures mean is 0 and variance 1 but does ensure any particular minimum or maximum.

- **Robust Scaling** works like Standardization but uses the median and quartiles.

- **Normalization** scales each data point so that feature vector has Euclidean length of 1. Useful when direction but not magnitude matters.

- **Custom** transformation apply a user-specified function to selected variables.
Scaling Techniques

Original Data

StandardScaler

MinMaxScaler

RobustScaler

Normalizer
We first fit the scaler to the training data.

```python
In [4]: from sklearn.preprocessing import MinMaxScaler
In [5]: scaler = MinMaxScaler()
In [6]: scaler.fit(X_train)
Out[6]: MinMaxScaler(copy=True, feature_range=(0, 1))
```

We do not use the test data when fitting a scaler, as that will leak test data into the model building process, resulting in inflated model performance when measured on the test data set.
Scaling with Scikit-learn

We scale the training data with the `transform()` method.

```python
In [28]: X_train_scaled = scaler.transform(X_train)
```

We verify that the maximum values are now 1.

```python
In [30]: X_train.max(axis=0)
Out[30]:
array([2000, 29,  5.040929, 1999.58055556, 1., 1., 1.])
In [31]: X_train_scaled.max(axis=0)
Out[31]: array([1., 1., 1., 1., 1., 1., 1.])
```

We verify that the minimum values are now 0.

```python
In [33]: X_train.min(axis=0)
Out[33]: array([1., 1., 0.4212135, 0., 0., 0., 0.])
In [34]: X_train_scaled.min(axis=0)
Out[34]: array([0., 0., 0., 0., 0., 0., 0.])
```
We first fit the scaler to the training data using the same transformation.

```python
In [36]: X_test_scaled = scaler.transform(X_test)
In [37]: X_test_scaled.min(axis=0)
Out[37]:
array([0., 0., 0.47081135, 0., 0., 0., 0.])
In [38]: X_test_scaled.max(axis=0)
Out[38]:
array([1., 0.32142857, 1., 0.99455823, 1., 1., 1.])
```

Since we used the training data to transform, scaled test mins and maxs are not always 0 or 1 respectively.
Custom Transformations

Transforming skewed data using a function like log can make patterns easier to find.

We use the function

\[ \log(1 + x) \]

to transform data in this example.

In [40]: from sklearn.preprocessing import FunctionTransformer
In [41]: transformer = FunctionTransformer(np.log1p)
In [42]: X = np.array([[0, 1], [2, 3]])
In [43]: transformer.transform(X)
Out[43]:
array([[0., 0.69314718],
       [1.09861229, 1.38629436]])
References


3. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer. 2014.
