18. Machine Learning - Supervised Learning with Regressions and Classification Algorithms

3ikakke

Monday 2nd May 2022

Outline

- Objectives
- Data Preprocessing
- Dummy variables
- Scaling numeric data
- Selecting predictors for Multilinear and Logistic Regressions
- Splitting data into training and test (possibly validation)
- Regressions
- Classification
- Evaluating models
- Q&A
- Review of objectives
- Gist for the day

Objectives

- Understand preprocessing
- Splitting data into training and test (possibly validation)
- Selecting predictors for Multilinear regression & Logistic regression
- Understanding Regressions
- Understanding Classification
- Understanding how to evaluate models

Data Preprocessing - Missingness

- Exploring for missingness
  - Check for missingness and percentage missingness to determine if to keep predictors or discard them

```
print(dataset.isnull().sum())
#or visually
plt.figure(figsize=(16, 8))
sns.heatmap(dataset.isnull())
plt.show()
```

  - For missingness thats large (a good rule may be 10% or more), you may consider droping

```
dataset.drop(['var_1', 'var_2', ...'var_n'], axis=1)
#or
dataset.drop(['var_1', 'var_2', ...'var_n'], axis='columns')
```

  - For smaller missingness (5% and less) you may excude the missing observations or do simple imputation using mean values or a missing level within categories

```
#exclude missing
dataset = dataset.loc[~dataset['var'].isna()]
#impute for missing - note this is for numeric variables
dataset.loc[dataset['var'].isna(), 'var'] = dataset['var'].mean()
#impute for missing - note this is for categorical variables
dataset.loc[dataset['var'].isna(), 'var'] = 'unknown'
```

Data Preprocessing (contd) - Relationships

- Exploring for relationships to determine features
  - Bivariate analysis (correlations, compare means, compare frequencies)

```
#to get R
r = dataset.corr()
#to get R Squared
r2 = np.square(r)
#visualize
plt.figure(figsize=c(16, 4))
sns.heatmap(r2, cmap=sns.cm.rocket_1)
plt.show()
```

Data Preprocessing (contd) - Hypothesis testing

- Exploring for relationships to determine features (contd)
- Hypothesis testing (ttest, anova, chisquare tests, regression).

```
catvars = dataset.drop('label', axis=1).select_dtypes(exclude='number').columns
for var in catvars:
    crosstab = pd.crosstab(dataset[var, 'label'])
    x2, pvalue, dof, ev = chi2_contingency(crosstab)
    if pvalue < 0.05:
        print(var)
```

Data Preprocessing (contd) - Visualization

- Exploring for relationships to determine features (contd)
- Bivariate analysis visualizations (categorical vs numeric)

```
#for numeric labels
sns.pairplot(dataset)
#for categorical labels
sns.pairplot(dataset, hue='label')
```

- For comparing categorical variables with categorical outcomes the best approach will be build bivariate plots for each

```
catvars = dataset.drop('label', axis=1).select_dtypes(exclude='number').columns #to get the categor
for var in catvars:
    sns.countplot(data=dataset, x=var, hue='label')
    plt.show()
```

Dummy variables

- Categorical variables cannot be included in mathematical models hence need to be converted to numeric variables

```
catvars = dataset.select_dtypes(exclude='number').columns
dummies = pd.get_dummies(dataset[catvars], drop_first=True)
```

Scaling numeric data

- If the data variables or on very different scales you may consider scaling so everything is on the same scale

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(dataset.drop('label', axis=1))
scaled = scaler.transform(dataset.drop('label', axis=1))
scaled = pd.DataFrame(scaled)
dcolumns = dataset.drop('label', axis=1).columns
scaled.columns = dcolumns
```

Selecting predictors for Multilinear and Logistic Regressions

- Multi linear regresssion
  - Use the statsmodels module to build a model and explore how predictive each variable is before deciding on your fetures for your model building
  - Begin with a full model

  ```
  from statsmodels.formula.api import ols
  model = ols("label ~ feature_1 + feature_2 .... + feature_n", data=dataset).fit()
  print(model.summary())
  ```

  - Take out a feature at a time till you have all features with p-values under a threshold (usually not greater than 0.2)
- Logistic regression

  - Similar to multilinear regression
  - Begin with a full model

```
from statsmodels.formula.api import logit
model = logit("label ~ feature_1 + feature_2 .... + feature_n", data=dataset).fit()
print(model.summary())
```

  - Take out a feature at a time till you have all features with p-values under a threshold (usually not greater than 0.2)

Spliting data into training and test (possibly validation)

- From your analytical dataset create a variable for features and another for labels
- import the splitter
- create the split

```
from sklearn.model_select import train_test_split
label = dataset['label']
features = dataset.drop('label', axis=1)
training_features, test_features, training_labels, test_labels = train_test_split(features, label, test_
```

Regressions

- Setting up for regressions
  - import the algorithm from the family

```
from sklearn.linear_model import LinearRegression
```

  - initialize the algorithm

```
algo = LinearRegression()
```

- Simple linear regressions
  - This involves only 2 variables - single label and single feature

```
#train
algo.fit(training_features, training_labels)
predicted_labels = algo.predict(test_features)
```

- Multi linear regression
  - This involves more than 1 variable in the features
- Polynomial regressions
  - This involves an additional step of creating exponents of the variable with a curved relationship

```
dataset['var2'] = dataset['var'] ** 2
```

Understanding Classification (Logistic Regression)

- Logistic regregression
  - The label must be binary and converted to a dumy variable with 1 and 0 as only values

```
from sklearn.linear_model import LogisticRegression
algo = LogisticRegression()
```

```
algo.fit(training_features, training_labels)
predicted_labels = algo.predict(test_features)
```

Understanding Classification (KNN)

- K-Nearest Neighbor (KNN)
  - These can deal with variables with more than 2 categories
  - The labels dont need to be converted to numeric variables
  - The K-value needs to be set

```
from sklearn.neighbor import KNearestNeighbor
algo = KNearestNeighbor(n_neighbor=3)
algo.fit(training_features, training_labels)
predicted_labels = algo.predict(test_features)
```

Understanding Classification (Decision Tree)

- Decision Tree Classification
  - These can deal with variables with more than 2 categories
  - The labels dont need to be converted to numeric variables
  - These use entropy to determine the classes that exist within data

```
from sklearn.tree import DecisionTreeClassifier
algo = DecisionTreeClassifier()
algo.fit(training_features, training_labels)
predicted_labels = algo.predict(test_features)
```

Understanding Classification (Support Vector)

- Support Vector Machines
  - These can deal with variables with more than 2 categories
  - The labels dont need to be converted to numeric variables
  - These use datapoints to create rails to determine where new points belong
  - They operate on higher dimensions also

```
from sklearn.svm import SVC
algo = SVC()
algo.fit(training_features, training_labels)
predicted_labels = algo.predict(test_features)
```

Evaluating Regresssion Models

- SKLearn comes with a module for evaluation models called the metrics
- Regression assessment

- Mean Absolute Error

- Mean Square Error

- Root Mean Square Error

```
from sklearn.metrics import mean_absolute_error, mean_square_error
mae = mean_absolute_error(predicted_labels, test_labels) #Mean Absolute Error
mse = mean_square_error(predicted_labels, test_labels) #Mean Square Error
rmse = np.sqrt(mse) #Root Mean Square Error

print(mae)
print(mse)
print(rmse)
```

Evaluating Classification Models

- Classification assessment (Confusion Matrix)
  - Accuracy (correct predictions/all predictions)

  - Precision ($P(Actual|Predicted)$)

  - Recall ($P(Predicted|Actual)$)

  - F1-Score (Harmonic Mean of Precision and Recall $= 2 * \frac{precision*Recall}{precision+recall}$)

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(predicted_labels, test_labels))
print(confusion_matrix(predicted_labels, test_labels))
```

Q&A

Review of objectives

- Understand preprocessing

- Splitting data into training and test (possibly validation)
- Selecting predictors for Multilinear regression & Logistic regression

- Understanding Regressions

- Understanding Classification

- Understanding how to evaluate models

Gist for the day

- Get the gist from here
  - Pre Processing

  - Stepwise building of models for linear and logistic regrerssion

- Split data into training and test sets
- Regression algorithms
- Logistic Regression
- Classification Algorithms
- Evaluating Regression Models
- Evaluating Classification Models

- Get the pdf version from here
- The Jupyter Notebook will be added

Thanks for being active!