QBUS6850: Tutorial 4 - Linear Regression 2, Feature Extraction/Preparation, Logistic Regression

Objectives

- To know how to do model selection;
- To learn how to extract features;
- To learn how to do logistic regression;

1.1 Ordinary Least Squares Drawbacks

$$min_{\beta} \frac{1}{2N} \|t - X\beta\|^2$$
$$\beta = (X^T X)^{-1} X^T t$$

OLS (Ordinary least squares) is an unbiased estimator by definition, however this does not mean that it always produces a good model estimation.

OLS can completely fail or produce poor results under relatively common conditions.

The shared point of failure among all these conditions is that $X^T X$ is singular and OLS estimates depend on$(X^T X)^{-1}$. When $X^T X$ is singular we cannot compute its inverse.

This happens frequently due to:

- collinearity i.e. predictors (features) are not independent
- *d* > *N* i.e. number of features exceeds number of observations

Let $M = X^T X \in \mathbb{R}^{d \times d}$ then $M_{ij} = dot(X_{(:,i)}, X_{(:,j)})$.

Or in other words each element of $\boldsymbol{M}$ encodes the similarity or distance from each feature vector to every other feature vector. When we have collinear features this can result in $rank(\boldsymbol{M}) < d$ meaning that $\boldsymbol{M}$ is singular and we cannot compute its inverse.

Moreover, OLS may be subject to outliers. This means that a single large outlier value can greatly affect our regression line. Therefore, we know OLS is low (0) bias but high variance. We might wish to sacrifice some bias to achieve lower variance.

1.2 Ridge Regression

Ridge regression addresses both issues of high variance and inevitability.

By reducing the magnitude of the coefficients by using the L2 norm we can reduce the coefficient variance and thus the variance of the estimator. Therefore, the ridge regression objective function is

$$min_\beta \frac{1}{2N}\|t - X\beta\|^2 + \frac{\lambda}{2N}\|\beta\|^2$$

where λ is a tuneable parameter and controls the strength of the penalty. Therefore, the solution is given by

$$\beta = (X^T X + \lambda I)^{-1} X^T t$$

By reducing the variance of the estimator, ridge regression tends to be more "robust" against outliers than OLS since we have a solution with greater bias and less variance. Conveniently this means that $\boldsymbol{X^T X}$ is then modified to $\boldsymbol{X^T X + \lambda I}$. In other words, we add a diagonal component to the matrix that we want to invert. This diagonal component is called the "ridge". This diagonal component increases the rank so that the matrix is no longer singular.

We can use cross validation to find the shrinkage parameter that we believe will generalise best to unseen data.

About the Data

The dataset is a collection of community and crime statistics from http://archive.ics.uci.edu/ml/datasets/communities+and+crime

```
from sklearn.linear_model import LinearRegression, RidgeCV
import numpy as np
import pandas as pd

from sklearn.model_selection
import train_test_split
from sklearn.metrics import mean_squared_error
#http://archive.ics.uci.edu/ml/datasets/communities+and+crime
crime = pd.read_csv('communities.csv', header=None, na_values=['?'])
```

Pre-processing steps

```
#Delete the first 5 columns
crime = crime.iloc[:, 5:]
# Remove rows with missing entries
crime.dropna(inplace=True)
```

Extract all the features available as our predictors and the response variable (number of violent crimes per capita)

```
# Get the features X and target/response y
X = crime.iloc[:, :-1]
t = crime.iloc[:, -1]
```

To evaluate model performance we will split the data into train and test sets.

Our procedure will follow: - Fit model on training set - Test performance via loss function values on test set

```
# Split data into train and test sets

X_train, X_test, t_train, t_test = train_test_split(X, t, random_state=1)
```

Fit OLS model and calculate loss values over test data

```
lm = LinearRegression()
lm.fit(X_train, t_train)
preds_ols = lm.predict(X_test)
print("OLS: {0}".format(mean_squared_error(t_test, preds_ols)/2))
```

Generate possible candidate sets for *l*, fit the ridge model using cross validation on training set to find optimal *l* and calculate mse over test set.

```
alpha_range = 10.**np.arange(-2,3)

rregcv = RidgeCV(normalize=True, scoring='neg_mean_squared_error',
alphas=alpha_range)

rregcv.fit(X_train, t_train)

preds_ridge = rregcv.predict(X_test)
print("RIDGE: {0}".format(mean_squared_error(t_test, preds_ridge)/2))
```

However RidgeCV can also be used without those parameters eg.

```
rregcv = RidgeCV()

rregcv.fit(X_train, t_train)

preds_ridge = rregcv.predict(X_test)
print("RIDGE: {0}".format(mean_squared_error(t_test, preds_ridge)/2))
```

1.3 Text Feature Extraction

- Bag of words
- Embedding

1.3.1 Bag of Word

Bag of words(1-gram) counts the words and keep the numbers and serve as the features.

3 steps need to be performed to get Bag of Word (BOW) features:

1. Tokenizing: Segment the corpus into "words" 3
2. Counting: Count the appearance frequecy of difference words
3. Normalizing

`CountVectorizer` from sklearn combine the tokenizing and counting.

```
from sklearn.feature_extraction.text import CountVectorizer vectorizer =
CountVectorizer()

vectorizer
```

```
corpus = [
      'This is the first document.',

      'This is the second second document.',

      'And the third one.',

      'Is this the first document?',

      ]

X_txt = vectorizer.fit_transform(corpus)

vectorizer.get_feature_names() == (
      ['and', 'document', 'first', 'is', 'one',

      'second', 'the', 'third', 'this'])

#X is the BOW feature of X

print(X_txt.toarray())

#the value is vocab id

print(vectorizer.vocabulary_)
#Unseen     words are ignore

vectorizer.transform(['Something completely new.']).toarray()
```

Bag of Words features cannot carputer local information. E.g. "believe or not" has the same features as "not or believe". Bi-gram preserve more local information, which regards 2 contagious words as one word in the vocabulary.
For example, "believe or", "or not", "not or" and "or believe" are counted. The feature extraction is shown in the code below.

```
bigram_vectorizer = CountVectorizer(ngram_range=(2, 2),
token_pattern=r'\b\w+\b', min_df=1)
analyze = bigram_vectorizer.build_analyzer()
print(analyze('believe or not a b c d e'))
```

```
#Extract the features

In [94]: X_txt_2 = bigram_vectorizer.fit_transform(corpus).toarray()

print(X_txt_2)
```

1.3.2 tf-idf

Some words has very high frequency (e.g. “the”, “a”, ”which”), therefore, carrying not much meaningful information about the actual contents of the document.
We need to compensate them to prevent the high-frequency shadowing other words. *td* $idf(t,d) = tf(t,d) \times idf(t)$

$$idf(t) = \log\left(\frac{1+n_d}{1+df(d,t)}\right) + 1$$

Where n_d is the number of document. $df(t)$ is the number of documents containing t.

Each row is normalized to have unit Euclidean norm:

```
from sklearn.feature_extraction.text import TfidfTransformer

transformer = TfidfTransformer(smooth_idf=False)

counts = [[3, 0, 1],

[2, 0, 0],

[3, 0, 0],

[4, 0, 0],

[3, 2, 0],

[3, 0, 2]

]

tfidf = transformer.fit_transform(counts)

print(tfidf)

print(tfidf.toarray())
```

An even more concise way to compute the tf-idf features. Combine counts and tf-idf

```
from sklearn.feature_extraction.text import TfidfVectorizer vectorizer =
TfidfVectorizer()
```

```
print(corpus)

X_txt_3 = vectorizer.fit_transform(corpus)

print(X_txt_3.toarray())
```

1.4 Feature Selection

- LASSO
- Elastic-Net
- Refer to lecture slides for their loss functions

We can take two approaches when using LASSO or Elastic-net for feature selection. We can softly regularise the coefficients until a sufficient number are set to 0. Fortunately using cross validation to optimise the regularisation parameter lambda (called alpha in sklearn) usually results in many of the features being ignored since their coefficient values are shrunk to 0.

Alternatively, you can set a threshold value for the coefficient and find a suitable regularisation parameter that meets this requirement.

We will take the path of cross validation.

Note that due to the shared L1/L2 regularisation of Elastic-Net it does not aggressively prune features like Lasso. However, in practice it often performs better when used for regression prediction.

NOTE: You can also use LASSO/Elastic-Net for regular regression tasks.

Fit the LASSO model. Performed a train/test split and used CV on the train set to determine optimal parameters.

```
from sklearn.linear_model import LassoCV, ElasticNetCV

X= crime.iloc[:, :-1]
t = crime.iloc[:, -1]
X_train, X_test, t_train, t_test = train_test_split(X, t, random_state=1)

lascv = LassoCV()
lascv.fit(X_train, t_train)

preds_lassocv = lascv.predict(X_test)
print("LASSO: {0}".format(mean_squared_error(t_test, preds_lassocv)/2))
print("LASSO Lambda: {0}".format(lascv.alpha_))
```

Fit the Elastic-Net model

```
elascv = ElasticNetCV(max_iter=10000)
elascv.fit(X_train, t_train)

preds_elascv = elascv.predict(X_test)
print("Elastic Net: {0}".format(mean_squared_error(t_test, preds_elascv)/2))
print("Elastic Net Lambda: {0}".format(elascv.alpha_))
```

Determine which columns were retained by each model

```
columns = X.columns.values
lasso_cols = columns[np.nonzero(lascv.coef_)] print("LASSO Features:
{0}".format(lasso_cols))
elas_cols = columns[np.nonzero(elascv.coef_)] print("Elastic Features:
{0}".format(elas_cols))
```

1.5 Regression vs Classification

• Logistic Regression

WARNING: Do not use regression for classification tasks.

In general it is ill advised to use linear regression for classification tasks. Regression learns a continuous output variable from a predefined linear (or higher order) model. It learns the parameters of this model to predict an output.

Classification on the other hand is not explicitly interested in the underlying generative process. Rather it is a higher abstraction. We are not interested in the specific value of something. Instead we want to assign each data vector to the most likely class.

Logistic regression provides us with two desirable properties: - the output of the logistic function is the direct probability of the data vector belonging to the success case

$$f(x, \beta) = \frac{1}{1 + e^{(\beta_0 + \beta_1 x)}}$$

the logistic function is non-linear and more flexible than a linear regression, which can improve classification accuracy and is often more robust to outliers.

About the dataset

The data shows credit card loan status for many accounts with three features: student, balance remaining and income.

```
from sklearn.linear_model import LogisticRegression

df = pd.read_csv('Default.csv')
```

Convert the student category column to Boolean values

```
df.student = np.where(df.student == 'Yes', 1, 0)
```

Use the balance feature and set the default status as the target value to predict. You could also use all available features if you believe that they are informative.

```
X = df[['balance']]

t = df[['default']]

X_train, X_val, t_train, t_val = train_test_split(X, t, test_size=0.3,
random_state=1
```

Fit the Logistic Regression model

```
log_res = LogisticRegression()

log_res.fit(X_train, t_train)
```

Predict probabilities of default.

`predict_proba()` returns the probabilities of an observation belonging to each class. This is computed from the logistic regression function (see above). `predict()` is dependent on `predict_proba()`. `predict()` returns the class assignment based on the proability and the decision boundary. In other words, predict returns the most likely class i.e. the class with greatest probability or probability > 50%.

```
prob = log_res.predict_proba(pd.DataFrame({'balance': [1200, 2500]}))
print(prob)

print("Probability of default with Balance of 1200: {0:.2f}%".format(prob[0,1] *
100)
print("Probability of default with Balance of 2500: {0:.2f}%".format(prob[1,1] *
100)

outcome = log_res.predict(pd.DataFrame({'balance': [1200, 2500]}))

print("Assigned class with Balance of 1200: {0}".format(outcome[0]))
print("Assigned class with Balance of 2500: {0}".format(outcome[1]))
```

We can evaluate classification accuracy using confusion matrix (to be explanined in week 4 lecture) and the classification report

```
from sklearn.metrics import confusion_matrix

pred_log = log_res.predict(X_val)

print(confusion_matrix(t_val,pred_log))
```

```
from sklearn.metrics import classification_report
print(classification_report(t_val, pred_log, digits=3))
```