COMP 347: Applied Machine Learning **Lecture Notes**

Regression Case Study

Robert Utterback *09/04/2022*

Contents


```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import sklearn
```

1 Motivation

1.1 Experience

- Best way to learn ML is to do it!
- And use real data!
- Take a look at the sources of real data the book provides.
- Kaggle can help you get started, although remember that even those data sets are cleaner than real-life, both in the sense of data and in the sense of having a clear goal.

1.2 Main Steps

Big picture
Get the data
Explore
Preprocess/prepare
Train a model
Fine-tune
Present your solution
Deploy: launch, monitor, maintain

2 Case Study: Housing Prices

- I'm going to do basically the same case study, but with a slightly different dataset, just to give you a slightly different perspective, but not so different than it's overwhelming (I hope).
- We work for a company that wants to predict housing prices for census "districts", given some other data about the districts

2.1 Big Picture

- What's the problem? (Experts estimate it, very slowly)
- How will this model be used? (In a downstream system.)
- Current solution? Is ML the right approach?
- Q: What *type* of ML is this? (supervised, unsupervised, reinforcement) A: supervised
- Q: Classification, regression, other? A: regression
- Q: batch learning or online? A: batch will be fine here (at least at first)
- What's our performance measure? Very important to think about first!

2.1.1 Performance Measures and Notation

- Book chooses RMSE:

$$\mathrm{RMSE}(\boldsymbol{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2}$$

  which is fine.

- Note notation is opposite from what I said was usual!
- m = number of examples, n is number of features
- $\boldsymbol{x}^{(i)}$ is the i^{th} instance (technically as a column vector), $y^{(i)}$ is its label
- $\boldsymbol{X}$ is a matrix of all instances
- h is prediction function (*hypothesis*).
- Predicted values are $\hat{y}^{(i)} = h(\boldsymbol{x}^{(i)})$
- Alternative would be MAE:

$$\mathrm{MAE}(\boldsymbol{X}, h) = \frac{1}{m}\sum_{i=1}^{m}|h(\boldsymbol{x}^{(i)} - y^{(i)}|$$

- RMSE is a bit sensitive to outliers: due to squared values, they will outweight things and possibly produce large errors, which the model might overreact to.
- RMSE is also known as the ℓ_2 **norm**, which is an important concept
- MAE, by using absolute value instead of squares, is the ℓ_1 norm
- In general, the ℓ_k norm, $\|\boldsymbol{v}\|_k$ is

$$\left(|v_0|^k + |v_1|^k + \cdots + |v_n|^k\right)^{\frac{1}{k}}$$

- ℓ_0 is a bit weird but gives the number of nonzero elements in the vector
- ℓ_∞ is the maximum absolute val in the vector
- Notice that as k gets bigger it focuses more and more on the large values.

2.1.2 Check your assumptions

- e.g., do you really need prices, or just price groups?

2.2 Get the data

- I'm basically going to skip this.
- He talks about some good stuff with automatically downloading and unzip it, but you should be able to read it just fine.

```
from sklearn.datasets import load_boston
dataset = load_boston()
```

2.3 Exploring

```
df = pd.DataFrame(dataset.data)
df.head()
```

- It's convenient to turn it into a dataframe, but actually it is a separate data structures.
- Also try `keys()`, `data.shape`, `feature_names`, `DESCR`, `target`, to explore...
- But now the columns are just numbers. Let's make them features

```
df.columns = dataset.feature_names
# or from scratch: pd.DataFrame(boston['data'], columns = boston['feature_names'])
df.head()
```

- Let's also add the target "PRICE" as a column to get a full look at the data

```
df['PRICE'] = dataset.target
```

- Now try out `.info()` and `.describe()`
- Most are numeric, which makes things easy to work with...

feature name	**description**
CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centres
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per $10,000
PTRATIO	pupil-teacher ratio by town
B	$1000(Bk - 0.63)^2$ where Bk is the proportion of black people by town
LSTAT	% lower status of the population
MEDV	Median value of owner-occupied homes in $1000's

- Note that CHAS is basically a categorical variable (do .value_counts())
- Note also B (old dataset) – want to be careful what you "learn" from this and especially how this is applied – don't perpetuate biases.
- RAD is numeric, but I wonder if you could treat it as categorical...
- Let's look at some histograms:

```
# %matplotlib inline
df.hist(bins='auto', figsize=(20,15));
```

2.4 Making a Test Set

- Don't look too much at the data before making a test set: you might 'overfit' to what you see
- In the book he makes a big deal about what happens if you update the dataset
- In reality, this usually isn't that big a deal, except for really important/large projects
- We'll typically just use sklearn:

```
from sklearn.model_selection import train_test_split

X = df.drop('PRICE', axis=1)
X_train, X_test, y_train, y_test = \
    train_test_split(X, df['PRICE'],
                     test_size=0.20, random_state=42)
```

- Although his code to do it manually is interesting, let's analyze it:

```
import numpy as np
np.random.seed(42)

# For illustration only. Sklearn has train_test_split()
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]

train_set, test_set = split_train_test(housing, 0.2)
```

2.5 Continue Exploring

- Look for some correlations:

```
df.corr()['PRICE'].sort_values(ascending=False)
```

- Then let's plot one of the features, average # rooms per dwelling, against price

```
plt.scatter(df.RM, df.PRICE)
plt.xlabel("Average number of rooms per dwelling (RM)")
plt.ylabel("Housing Price")
plt.title("Relationship between RM and Price")
plt.show()
```

- Pandas can do better and plot lots of things for us:

```
from pandas.plotting import scatter_matrix

features = ['PRICE', "RM", "ZN", "PTRATIO", "LSTAT"]
scatter_matrix(df[features], figsize=(12,8));
```

- Clearly some of these have some things going on
- Things to think about here: are some values getting "clipped"? Do we need to transform/scale the data? Do we need to remove weird samples or classes of samples (usually because they have "fake" or missing values)?

2.6 Training a Model

- At this point the books going into a fair amount of detail about preprocessing:
- handling missing values, categorical data, scaling, and pipelines
- I'm going to skip that stuff and go straight to the model
- We will come back to that stuff, either soon, or as we need it.
- The actual training of models is quite simple!

```
[10.96952405 19.41196567 23.06419602 12.1470648  18.3738116  25.24677946
 20.77024774 23.90932632  7.81713319 19.60988098]
477    12.0
15     19.9
332    19.4
423    13.4
19     18.2
325    24.6
335    21.1
56     24.7
437     8.7
409    27.5
Name: PRICE, dtype: float64
```

- Notice what we want is a diagonal, though as we increase price we're getting further away
- This is a linear model, so it has a coefficient and an intercept
- `lm.coef_` and `lm.intercept_`
- View them with:

```
# lm.coef_ and lm.intercept_
list(zip(X.columns, lm.coef_))
# or
pd.DataFrame(list(zip(X.columns, lm.coef_)),
    columns = ['features', 'estimated coefficient'])
```

```
   features  estimated coefficient
0      CRIM              -0.113056
1        ZN               0.030110
2     INDUS               0.040381
3      CHAS               2.784438
4       NOX             -17.202633
5        RM               4.438835
6       AGE              -0.006296
7       DIS              -1.447865
8       RAD               0.262430
9       TAX              -0.010647
10  PTRATIO              -0.915456
11        B               0.012351
12    LSTAT              -0.508571
```

2.7 Evaluating

- Can eval with MSE
- Can get from the model itself or calculate manually

```
pred_train = lm.predict(X_train)
pred_test = lm.predict(X_test)
mse_train = np.mean((y_train - pred_train) ** 2)
mse_test = np.mean((y_test - pred_test) ** 2)
print("Training MSE: {}".format(mse_train))
print("Testing MSE: {}".format(mse_test))
```

```
Training MSE: 21.641412753226316
Testing MSE: 24.291119474973385
```

- Note also from the book:

  ```
  from sklearn.metrics import mean_squared_error
  mse = mean_squared_error(y_train, pred_train)
  print(mse, np.sqrt(mse))
  ```

- Although this seems better than the book's California predictions, keep in mind (1) these prices are in 1000s of dollars, and (2) this was in the 70s – there has been a lot of inflation!
- You can visualize errors with a **residual plot**
- The residual is the different between target and prediction
- So you want it to however around zero

```
plt.scatter(pred_train, pred_train - y_train, c='b', alpha=.5, s=40)
plt.scatter(pred_test, pred_test - y_test, c='g', alpha=0.5, s=40)
plt.hlines(y=0, xmin=0, xmax=50)
```