ML Tutorial 4

June 25, 2018

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import sklearn
        import seaborn as sns

        %matplotlib inline
```

0.1 Machine Learning Algorithms

We can breakdown the problem of Machine Learning into two subtypes:

0.1.1 Regression

In general, these are problems where we are trying to predict a continuous variable. There are a few models that you can use to solve these type of problems:

- Ordinary Least Squares (Linear Regression)
- LASSO Regression
- Ridge Regression
- k nearest neighbours (kNN) Regression
- Elastic Net

You should already have seen and used the Linear Regression in practice. All subsequent models are similar in nature in the sense that we are trying to minimise the error function on the parameter space. I.e, find a set of $\boldsymbol{\beta}$ such that $\epsilon(\boldsymbol{\beta}) \geq 0$ is minimum. We saw that the error function for the **ordinary linear regression** is as follows:

$$\epsilon(\boldsymbol{\beta}) = MSE(\boldsymbol{\beta})$$

For the LASSO, Ridge and Elastic Net regression models, we are trying to minimise the following functions with respect to $\boldsymbol{\beta}$:

LASSO

$$\epsilon(\boldsymbol{\beta}) = MSE(\boldsymbol{\beta}) + \alpha \sum_{i=1}^{n} |\beta|$$

for some chosen α.

Ridge

$$\epsilon(\boldsymbol{\beta}) = MSE(\beta) + \alpha\boldsymbol{\beta}^\intercal\boldsymbol{\beta}$$

for some chosen α. This is also equivalent to:

$$\epsilon(\boldsymbol{\beta}) = MSE(\beta) + \alpha\sum_{i=1}^{n}\beta_i^2$$

0.1.2 Classification

These are problems where we the target variable is **categorical**. The following models can be used to solve these type of problems:

- Logistic Regression
- kNN Classification
- Decision Trees

Note that the models listed so far are by no means exhaustive.

0.1.3 Modelling on Iris

```
In [2]: iris = sns.load_dataset("iris")
```

```
In [3]: iris.head()
```

```
Out[3]:    sepal_length  sepal_width  petal_length  petal_width species
        0           5.1          3.5           1.4          0.2  setosa
        1           4.9          3.0           1.4          0.2  setosa
        2           4.7          3.2           1.3          0.2  setosa
        3           4.6          3.1           1.5          0.2  setosa
        4           5.0          3.6           1.4          0.2  setosa
```

```
In [4]: iris.columns
```

```
Out[4]: Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
               'species'],
              dtype='object')
```

Now, we implement **one hot encoding** in our feature matrix:

```
In [5]: iris["is_setosa"] = (iris['species'] == "setosa")*1
        iris["is_virginica"] = (iris['species'] == "virginica")*1
        iris["is_versicolor"] = (iris['species'] == "versicolor")*1
```

```
In [6]: iris.columns
```

```
Out[6]: Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species',
               'is_setosa', 'is_virginica', 'is_versicolor'],
              dtype='object')
```

We now split the data-set into our test and training sets containing 30% and 70% data respectively:

```
In [23]: n = np.int(0.3*iris.shape[0])
         idx = np.arange(0,iris.shape[0])
         np.random.shuffle(idx)
         iris_test = iris.iloc[idx[:n]]
         iris_train = iris.iloc[idx[n:]]
```

Say we want to predict the sepal width of the plant based on the other features minus the species. Let's consider the ridge regression model:

We first need to standardise the feature observations as the α weights each parameter equally.

```
In [32]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
```

Now, import the ridge regression model from the sklearn library:

```
In [52]: from sklearn.linear_model import Ridge
         ridge_model = Ridge(alpha=1.5)
```

Let's get started with data preprocessing.

We first drop the target column from our feature set, as well as our species column, and convert the dataframe into a NumPy array.

```
In [53]: x = iris_train.drop(['species','sepal_width'],axis = 1).values

In [54]: scaler.fit(x)
         x = scaler.transform(x)

In [55]: y = iris_train['sepal_width']

         ridge_model.fit(x,y)

Out[55]: Ridge(alpha=1.5, copy_X=True, fit_intercept=True, max_iter=None,
            normalize=False, random_state=None, solver='auto', tol=0.001)
```

Now we fit the model using the predictors from the test set, be sure to scale your predictors first!

```
In [56]: obs = iris_test['sepal_width']
         predictor = iris_test.drop(['species','sepal_width'],axis=1).values
         scaler.fit(predictor)
         predictor = scaler.transform(predictor)

In [57]: predicted = ridge_model.predict(predictor)

In [63]: MSE = ((obs - predicted)**2).mean()
         MSE
```

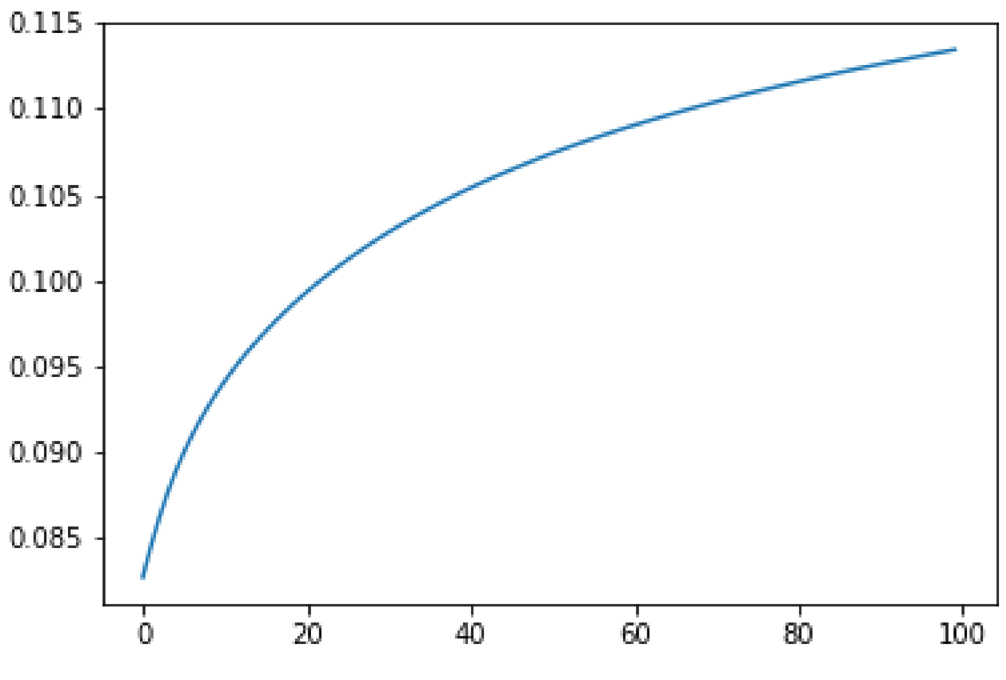
```
Out[63]: 0.08537061154862623
```

Now, try testing the model on a different α:

```
In [104]: def train_ridge(alpha):
              ridge_model = Ridge(alpha)
              x = iris_train.drop(['species','sepal_width'],axis = 1).values
              scaler.fit(x)
              x = scaler.transform(x)
              y = iris_train['sepal_width']
              ridge_model.fit(x,y)
              obs = iris_test['sepal_width']
              predictor = iris_test.drop(['species','sepal_width'],axis=1).values
              scaler.fit(predictor)
              predictor = scaler.transform(predictor)
              predicted = ridge_model.predict(predictor)
              MSE = ((obs - predicted)**2).mean()
              return MSE
```

```
In [105]: alpha_vec = pd.Series(np.arange(0.001,100,1))
          mse_vec = alpha_vec.apply(train_ridge)
          plt.plot(mse_vec)
```

```
Out[105]: [<matplotlib.lines.Line2D at 0x11117b208>]
```

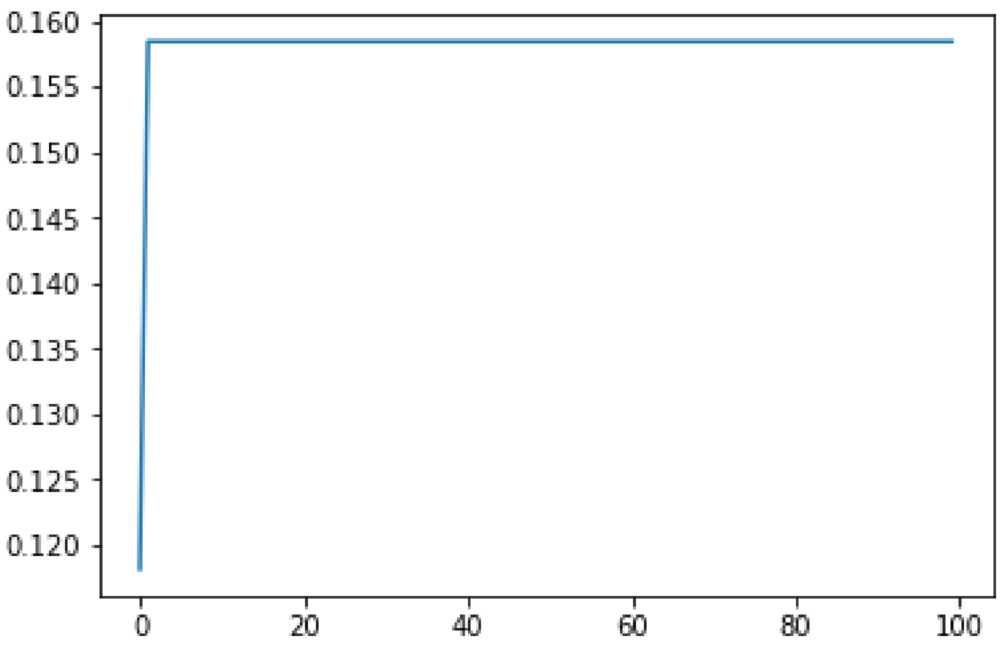


Let's try training a LASSO model on the data:

```
In [106]: from sklearn.linear_model import Lasso
In [107]: def train_LASSO(alpha):
              ridge_model = Lasso(alpha)
              x = iris_train.drop(['species','sepal_width'],axis = 1).values
              scaler.fit(x)
              x = scaler.transform(x)
              y = iris_train['sepal_width']
              ridge_model.fit(x,y)
              obs = iris_test['sepal_width']
              predictor = iris_test.drop(['species','sepal_width'],axis=1).values
              scaler.fit(predictor)
              predictor = scaler.transform(predictor)
              predicted = ridge_model.predict(predictor)
              MSE = ((obs - predicted)**2).mean()
              return MSE
```

```
In [108]: alpha_vec = pd.Series(np.arange(0.1,100,1))
          mse_vec = alpha_vec.apply(train_LASSO)
          plt.plot(mse_vec)
```

```
Out[108]: [<matplotlib.lines.Line2D at 0x1112915f8>]
```



0.1.4 Model Selection using Cross Validation

The idea of splitting the data into train and test sets extends to a routine where we can split the data up further so as to permutate what data the model can be used to train on. This is a heuristic method to allow us to be more confident of our model's prediction on general undiscovered data.

To do this, we split up the data set into k parts, where k is an integer. Then we train the model on k-1 parts of the data set and test on the remaining. We then iterate the training process with different sets of k-1 parts of the data and test on the remaining. We then calculate the average MSE from the process and call it the **k-fold cross validation error**. This can be used to determine model suitability.

In sklearn, we have a cross_validate object that will allow us to do cross validation easily. However, you need to be aware of the tendency of the cross_validation process to validate on a metric of its own choosing. This is why when we are comparing models from cross validation, we need to ensure that the scoring parameter is standard across all models.

```
In [101]: from sklearn.model_selection import cross_validate
          cv_results = cross_validate(ridge_model,x,y,cv= 5,return_train_score=True,scoring='n
```

To find the average mean squared error from the cross validation process, we have:

```
In [103]: cv_results['test_score'].mean()

Out[103]: -0.07103235052876726
```

In general, the higher the cross validated negative mean squared error, the better the model.

But what if we have many model parameters to consider? (recall the α parameter for the LASSO and Ridge models)

We can iterate through a list of α parameters and obtaining a CV error on each parameter:

```
In [118]: from sklearn.model_selection import GridSearchCV

          parameters = {
              'alpha': [0.1,1,3,5,10,100,101]

          }



          clf = GridSearchCV(Ridge(),parameters,scoring="neg_mean_squared_error")

          clf.fit(x,y)

Out[118]: GridSearchCV(cv=None, error_score='raise',
                 estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
             normalize=False, random_state=None, solver='auto', tol=0.001),
                 fit_params=None, iid=True, n_jobs=1,
                 param_grid={'alpha': [0.1, 1, 3, 5, 10, 100, 101]},
                 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                 scoring='neg_mean_squared_error', verbose=0)
```

Then we can get the parameter with which the mean test score (the CV error) is minimal.

```
In [126]: mean_test_score = clf.cv_results_['mean_test_score']
          np.argmax(mean_test_score)

Out[126]: 1
```

```
In [124]: mean_test_score[1]

Out[124]: -0.07023251758027858

In [125]: parameters['alpha'][1]

Out[125]: 1
```

Thus we see that AMONGST the alpha parameters we tested on, $\alpha = 1$ gives the best CV error.