Week 5 - Neural Networks - Estimation - Python

October 10, 2018

1 Data Warehousing and Data Mining

1.1 Labs

1.1.1 Prepared by Gilroy Gordon

Contact Information SCIT ext. 3643
ggordonutech@gmail.com
gilroy.gordon@utech.edu.jm

1.1.2 Week 5 - Neural Networks in Python

Additional Reference Resources:
http://scikit-learn.org/stable/modules/neural_networks_supervised.html

1.2 Objectives

```
> Data Preprocessing
     > Min Max Scaling
> Data Transformation
> Data Mining
     > Neural Networks (Classification and Estimation)
> Model Evaluation and Prediction
     > Train/Test Split - 70/30
> Presentation
     > Plots
```

1.3 Import required libraries and acquire data

```
In [1]: # import required libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [2]: data_path = './data/hr_data.csv' # Path to data file
        data = pd.read_csv(data_path)
        data.head(15)
```

```
Out[2]:     satisfaction_level  last_evaluation  number_project  average_montly_hours  \
        0                 0.38             0.53               2                   157
        1                 0.80             0.86               5                   262
        2                 0.11             0.88               7                   272
        3                 0.72             0.87               5                   223
        4                 0.37             0.52               2                   159
        5                 0.41             0.50               2                   153
        6                 0.10             0.77               6                   247
        7                 0.92             0.85               5                   259
        8                 0.89             1.00               5                   224
        9                 0.42             0.53               2                   142
        10                0.45             0.54               2                   135
        11                0.11             0.81               6                   305
        12                0.84             0.92               4                   234
        13                0.41             0.55               2                   148
        14                0.36             0.56               2                   137

            time_spend_company  Work_accident  left  promotion_last_5years  sales  \
        0                    3              0     1                      0  sales
        1                    6              0     1                      0  sales
        2                    4              0     1                      0  sales
        3                    5              0     1                      0  sales
        4                    3              0     1                      0  sales
        5                    3              0     1                      0  sales
        6                    4              0     1                      0  sales
        7                    5              0     1                      0  sales
        8                    5              0     1                      0  sales
        9                    3              0     1                      0  sales
        10                   3              0     1                      0  sales
        11                   4              0     1                      0  sales
        12                   5              0     1                      0  sales
        13                   3              0     1                      0  sales
        14                   3              0     1                      0  sales

            salary
        0      low
        1   medium
        2   medium
        3      low
        4      low
        5      low
        6      low
        7      low
        8      low
```

```
9      low
10     low
11     low
12     low
13     low
14     low
```

```
In [3]: # What columns are in the data set ? Do they have spaces that I should consider
        data.columns
```

```
Out[3]: Index(['satisfaction_level', 'last_evaluation', 'number_project',
               'average_montly_hours', 'time_spend_company', 'Work_accident', 'left',
               'promotion_last_5years', 'sales', 'salary'],
              dtype='object')
```

1.4 Aim: Can we determine a person's `Satisfaction Level` based on the other factors?

satisfaction_level = a(last_evaluation) + b(number_project) + c(average_montly_hours) + d(time_spend_company)

The coefficients a-d, what are they? What is the relationship between the variables? Does multicolinearity exist?

I have created a function below `create_label_encoder_dict` to assist with this. The function accepts a dataframe object and uses the `LabelEncoder` class from `sklearn.preprocessing` to encode (dummy encoding) or transform non-numerical columns to numbers. Finally it returns a dictionary object of all the encoders created for each column.

The LabelEncoder is a useful resource as it not only automatically transforms all values in a column but also keeps a track of what values were transformed from. i.e. It will change all `Female` to `0` and all `Male` to `1`

```
In [4]: def create_label_encoder_dict(df):
            from sklearn.preprocessing import LabelEncoder

            label_encoder_dict = {}
            for column in df.columns:
                # Only create encoder for categorical data types
                if not np.issubdtype(df[column].dtype, np.number) and column != 'Age':
                    label_encoder_dict[column]= LabelEncoder().fit(df[column])
            return label_encoder_dict
```

```
In [5]: label_encoders = create_label_encoder_dict(data)
        print("Encoded Values for each Label")
        print("="*32)
        for column in label_encoders:
            print("="*32)
            print('Encoder(%s) = %s' % (column, label_encoders[column].classes_ ))
            print(pd.DataFrame([range(0,len(label_encoders[column].classes_))], columns=label_en
```

```
Encoded Values for each Label
================================
================================
Encoder(sales) = ['IT' 'RandD' 'accounting' 'hr' 'management' 'marketing' 'product_mng'
 'sales' 'support' 'technical']
             Encoded Values
IT                        0
RandD                     1
accounting                2
hr                        3
management                4
marketing                 5
product_mng               6
sales                     7
support                   8
technical                 9
================================
Encoder(salary) = ['high' 'low' 'medium']
        Encoded Values
high                 0
low                  1
medium               2
```

In [6]:
```
# Apply each encoder to the data set to obtain transformed values
data2 = data.copy() # create copy of initial data set
for column in data2.columns:
    if column in label_encoders:
        data2[column] = label_encoders[column].transform(data2[column])

print("Transformed data set")
print("="*32)
data2.head(15)
```

```
Transformed data set
================================
```

Out[6]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	\
0	0.38	0.53	2	157	
1	0.80	0.86	5	262	
2	0.11	0.88	7	272	
3	0.72	0.87	5	223	
4	0.37	0.52	2	159	
5	0.41	0.50	2	153	
6	0.10	0.77	6	247	
7	0.92	0.85	5	259	
8	0.89	1.00	5	224	

```
9                 0.42             0.53                2                   142
10                0.45             0.54                2                   135
11                0.11             0.81                6                   305
12                0.84             0.92                4                   234
13                0.41             0.55                2                   148
14                0.36             0.56                2                   137

    time_spend_company  Work_accident  left  promotion_last_5years  sales  \
0                    3              0     1                      0      7
1                    6              0     1                      0      7
2                    4              0     1                      0      7
3                    5              0     1                      0      7
4                    3              0     1                      0      7
5                    3              0     1                      0      7
6                    4              0     1                      0      7
7                    5              0     1                      0      7
8                    5              0     1                      0      7
9                    3              0     1                      0      7
10                   3              0     1                      0      7
11                   4              0     1                      0      7
12                   5              0     1                      0      7
13                   3              0     1                      0      7
14                   3              0     1                      0      7

    salary
0        1
1        2
2        2
3        1
4        1
5        1
6        1
7        1
8        1
9        1
10       1
11       1
12       1
13       1
14       1
```

In [7]:
```
# separate our data into dependent (Y) and independent(X) variables
X_data = data2[['last_evaluation','number_project','average_montly_hours','time_spend_co
Y_data = data2['satisfaction_level']
```

1.5 70/30 Train Test Split

We will split the data using a 70/30 split. i.e. 70% of the data will be randomly chosen to train the model and 30% will be used to evaluate the model

```
In [8]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X_data, Y_data, test_size=0.30)
```

```
In [9]: from sklearn.neural_network import MLPClassifier, MLPRegressor
```

```
In [10]: # Create an instance of linear regression
         reg = MLPRegressor()
```

```
In [11]: reg.fit(X_train,y_train)
```

```
Out[11]: MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(100,), learning_rate='constant',
                learning_rate_init=0.001, max_iter=200, momentum=0.9,
                nesterovs_momentum=True, power_t=0.5, random_state=None,
                shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
                verbose=False, warm_start=False)
```

```
In [12]: help(MLPRegressor)
```

```
Help on class MLPRegressor in module sklearn.neural_network.multilayer_perceptron:

class MLPRegressor(BaseMultilayerPerceptron, sklearn.base.RegressorMixin)
 |  Multi-layer Perceptron regressor.
 |
 |  This model optimizes the squared-loss using LBFGS or stochastic gradient
 |  descent.
 |
 |  .. versionadded:: 0.18
 |
 |  Parameters
 |  ----------
 |  hidden_layer_sizes : tuple, length = n_layers - 2, default (100,)
 |      The ith element represents the number of neurons in the ith
 |      hidden layer.
 |
 |  activation : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'
 |      Activation function for the hidden layer.
 |
 |      - 'identity', no-op activation, useful to implement linear bottleneck,
 |        returns f(x) = x
 |
 |      - 'logistic', the logistic sigmoid function,
 |        returns f(x) = 1 / (1 + exp(-x)).
```

```
 |
 |      - 'tanh', the hyperbolic tan function,
 |        returns f(x) = tanh(x).
 |
 |      - 'relu', the rectified linear unit function,
 |        returns f(x) = max(0, x)
 |
 |  solver : {'lbfgs', 'sgd', 'adam'}, default 'adam'
 |      The solver for weight optimization.
 |
 |      - 'lbfgs' is an optimizer in the family of quasi-Newton methods.
 |
 |      - 'sgd' refers to stochastic gradient descent.
 |
 |      - 'adam' refers to a stochastic gradient-based optimizer proposed by
 |        Kingma, Diederik, and Jimmy Ba
 |
 |      Note: The default solver 'adam' works pretty well on relatively
 |      large datasets (with thousands of training samples or more) in terms of
 |      both training time and validation score.
 |      For small datasets, however, 'lbfgs' can converge faster and perform
 |      better.
 |
 |  alpha : float, optional, default 0.0001
 |      L2 penalty (regularization term) parameter.
 |
 |  batch_size : int, optional, default 'auto'
 |      Size of minibatches for stochastic optimizers.
 |      If the solver is 'lbfgs', the classifier will not use minibatch.
 |      When set to "auto", `batch_size=min(200, n_samples)`
 |
 |  learning_rate : {'constant', 'invscaling', 'adaptive'}, default 'constant'
 |      Learning rate schedule for weight updates.
 |
 |      - 'constant' is a constant learning rate given by
 |        'learning_rate_init'.
 |
 |      - 'invscaling' gradually decreases the learning rate ``learning_rate_``
 |        at each time step 't' using an inverse scaling exponent of 'power_t'.
 |        effective_learning_rate = learning_rate_init / pow(t, power_t)
 |
 |      - 'adaptive' keeps the learning rate constant to
 |        'learning_rate_init' as long as training loss keeps decreasing.
 |        Each time two consecutive epochs fail to decrease training loss by at
 |        least tol, or fail to increase validation score by at least tol if
 |        'early_stopping' is on, the current learning rate is divided by 5.
 |
 |      Only used when solver='sgd'.
```

```
 |
 |  learning_rate_init : double, optional, default 0.001
 |      The initial learning rate used. It controls the step-size
 |      in updating the weights. Only used when solver='sgd' or 'adam'.
 |
 |  power_t : double, optional, default 0.5
 |      The exponent for inverse scaling learning rate.
 |      It is used in updating effective learning rate when the learning_rate
 |      is set to 'invscaling'. Only used when solver='sgd'.
 |
 |  max_iter : int, optional, default 200
 |      Maximum number of iterations. The solver iterates until convergence
 |      (determined by 'tol') or this number of iterations. For stochastic
 |      solvers ('sgd', 'adam'), note that this determines the number of epochs
 |      (how many times each data point will be used), not the number of
 |      gradient steps.
 |
 |  shuffle : bool, optional, default True
 |      Whether to shuffle samples in each iteration. Only used when
 |      solver='sgd' or 'adam'.
 |
 |  random_state : int, RandomState instance or None, optional, default None
 |      If int, random_state is the seed used by the random number generator;
 |      If RandomState instance, random_state is the random number generator;
 |      If None, the random number generator is the RandomState instance used
 |      by `np.random`.
 |
 |  tol : float, optional, default 1e-4
 |      Tolerance for the optimization. When the loss or score is not improving
 |      by at least tol for two consecutive iterations, unless `learning_rate`
 |      is set to 'adaptive', convergence is considered to be reached and
 |      training stops.
 |
 |  verbose : bool, optional, default False
 |      Whether to print progress messages to stdout.
 |
 |  warm_start : bool, optional, default False
 |      When set to True, reuse the solution of the previous
 |      call to fit as initialization, otherwise, just erase the
 |      previous solution.
 |
 |  momentum : float, default 0.9
 |      Momentum for gradient descent update.  Should be between 0 and 1. Only
 |      used when solver='sgd'.
 |
 |  nesterovs_momentum : boolean, default True
 |      Whether to use Nesterov's momentum. Only used when solver='sgd' and
 |      momentum > 0.
```

```
|
|  early_stopping : bool, default False
|      Whether to use early stopping to terminate training when validation
|      score is not improving. If set to true, it will automatically set
|      aside 10% of training data as validation and terminate training when
|      validation score is not improving by at least tol for two consecutive
|      epochs.
|      Only effective when solver='sgd' or 'adam'
|
|  validation_fraction : float, optional, default 0.1
|      The proportion of training data to set aside as validation set for
|      early stopping. Must be between 0 and 1.
|      Only used if early_stopping is True
|
|  beta_1 : float, optional, default 0.9
|      Exponential decay rate for estimates of first moment vector in adam,
|      should be in [0, 1). Only used when solver='adam'
|
|  beta_2 : float, optional, default 0.999
|      Exponential decay rate for estimates of second moment vector in adam,
|      should be in [0, 1). Only used when solver='adam'
|
|  epsilon : float, optional, default 1e-8
|      Value for numerical stability in adam. Only used when solver='adam'
|
|  Attributes
|  ----------
|  loss_ : float
|      The current loss computed with the loss function.
|
|  coefs_ : list, length n_layers - 1
|      The ith element in the list represents the weight matrix corresponding
|      to layer i.
|
|  intercepts_ : list, length n_layers - 1
|      The ith element in the list represents the bias vector corresponding to
|      layer i + 1.
|
|  n_iter_ : int,
|      The number of iterations the solver has ran.
|
|  n_layers_ : int
|      Number of layers.
|
|  n_outputs_ : int
|      Number of outputs.
|
|  out_activation_ : string
```

```
 |      Name of the output activation function.
 |
 |  Notes
 |  -----
 |  MLPRegressor trains iteratively since at each time step
 |  the partial derivatives of the loss function with respect to the model
 |  parameters are computed to update the parameters.
 |
 |  It can also have a regularization term added to the loss function
 |  that shrinks model parameters to prevent overfitting.
 |
 |  This implementation works with data represented as dense and sparse numpy
 |  arrays of floating point values.
 |
 |  References
 |  ----------
 |  Hinton, Geoffrey E.
 |      "Connectionist learning procedures." Artificial intelligence 40.1
 |      (1989): 185-234.
 |
 |  Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of
 |      training deep feedforward neural networks." International Conference
 |      on Artificial Intelligence and Statistics. 2010.
 |
 |  He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level
 |      performance on imagenet classification." arXiv preprint
 |      arXiv:1502.01852 (2015).
 |
 |  Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic
 |      optimization." arXiv preprint arXiv:1412.6980 (2014).
 |
 |  Method resolution order:
 |      MLPRegressor
 |      BaseMultilayerPerceptron
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.base.RegressorMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, ba
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  predict(self, X)
 |      Predict using the multi-layer perceptron model.
 |
 |      Parameters
```

```
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          The input data.
 |
 |      Returns
 |      -------
 |      y : array-like, shape (n_samples, n_outputs)
 |          The predicted values.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseMultilayerPerceptron:
 |
 |  fit(self, X, y)
 |      Fit the model to data matrix X and target(s) y.
 |
 |      Parameters
 |      ----------
 |      X : array-like or sparse matrix, shape (n_samples, n_features)
 |          The input data.
 |
 |      y : array-like, shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels in classification, real numbers in
 |          regression).
 |
 |      Returns
 |      -------
 |      self : returns a trained MLP model.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseMultilayerPerceptron:
 |
 |  partial_fit
 |      Fit the model to data matrix X and target y.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          The input data.
 |
 |      y : array-like, shape (n_samples,)
 |          The target values.
 |
 |      Returns
```

```
 |      -------
 |      self : returns a trained MLP model.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : boolean, optional
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : mapping of string to any
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as pipelines). The latter have parameters of the form
 |      ``<component>__<parameter>`` so that it's possible to update each
 |      component of a nested object.
 |
 |      Returns
 |      -------
 |      self
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.BaseEstimator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
```

```
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.RegressorMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Returns the coefficient of determination R^2 of the prediction.
 |
 |      The coefficient R^2 is defined as (1 - u/v), where u is the residual
 |      sum of squares ((y_true - y_pred) ** 2).sum() and v is the total
 |      sum of squares ((y_true - y_true.mean()) ** 2).sum().
 |      The best possible score is 1.0 and it can be negative (because the
 |      model can be arbitrarily worse). A constant model that always
 |      predicts the expected value of y, disregarding the input features,
 |      would get a R^2 score of 0.0.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape = (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
 |          True values for X.
 |
 |      sample_weight : array-like, shape = [n_samples], optional
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          R^2 of self.predict(X) wrt. y.
```

In [18]:
```
reg.n_layers_ # Number of layers utilized
```

Out[18]:
```
3
```

In [20]:
```
# Make predictions using the testing set
test_predicted = reg.predict(X_test)
test_predicted
```

Out[20]:
```
array([0.66908448, 0.65429664, 0.60327367, ..., 0.47367503, 0.72666922,
       0.57838912])
```

In [21]:
```
data3 = X_test.copy()
data3['predicted_satisfaction_level']=test_predicted
data3['satisfaction_level']=y_test
data3.head()
```

Out[21]:
```
      last_evaluation  number_project  average_montly_hours  \
6335             0.81               5                   143
```

```
7020            0.73             3                   138
8864            0.56             3                   142
7454            0.57             4                   141
13361           0.61             4                   118

       time_spend_company  predicted_satisfaction_level  satisfaction_level
6335                    2                      0.669084                0.80
7020                    3                      0.654297                0.62
8864                    3                      0.603274                0.81
7454                    3                      0.577754                0.49
13361                   5                      0.541316                0.54
```

In [23]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [24]:
```
# The mean squared error don't worry guys we can do this
print("Mean squared error: %.2f" % mean_squared_error(y_test, test_predicted))
```

```
Mean squared error: 0.06
```

In [25]:
```
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, test_predicted))
```

```
Variance score: 0.07
```

2 Preprocessing

In [30]:
```
def create_min_max_scaler_dict(df):
    from sklearn.preprocessing import MinMaxScaler
    min_max_scaler_dict = {}
    for column in df.columns:
        # Only create encoder for categorical data types
        if np.issubdtype(df[column].dtype, np.number):
            min_max_scaler_dict[column]= MinMaxScaler().fit(pd.DataFrame(df[column]))
    return min_max_scaler_dict
```

In [31]:
```
min_max_scalers = create_min_max_scaler_dict(data)
print("Min Max Values for each Label")
print("="*32)
min_max_scalers
```

```
Min Max Values for each Label
================================
```

Out[31]:
```
{'Work_accident': MinMaxScaler(copy=True, feature_range=(0, 1)),
 'average_montly_hours': MinMaxScaler(copy=True, feature_range=(0, 1)),
 'last_evaluation': MinMaxScaler(copy=True, feature_range=(0, 1)),
```

```
 'left': MinMaxScaler(copy=True, feature_range=(0, 1)),
 'number_project': MinMaxScaler(copy=True, feature_range=(0, 1)),
 'promotion_last_5years': MinMaxScaler(copy=True, feature_range=(0, 1)),
 'satisfaction_level': MinMaxScaler(copy=True, feature_range=(0, 1)),
 'time_spend_company': MinMaxScaler(copy=True, feature_range=(0, 1))}
```

```
In [48]: #retrieving a scacler
         time_spend_company_scaler=min_max_scalers['time_spend_company']

In [49]: time_spend_company_scaler

Out[49]: MinMaxScaler(copy=True, feature_range=(0, 1))

In [50]: time_spend_company_scaler.data_max_ #Maximum value

Out[50]: array([10.])

In [51]: time_spend_company_scaler.data_min_ # Minimum value

Out[51]: array([2.])

In [52]: time_spend_company_scaler.data_range_ # Range = Max- Min

Out[52]: array([8.])

In [53]: pd.DataFrame([
             {
                 'column':col,
                 'min':min_max_scalers[col].data_min_[0],
                 'max':min_max_scalers[col].data_max_[0],
                 'range':min_max_scalers[col].data_range_[0] }  for col in min_max_scalers])
```

Out[53]:

	column	max	min	range
0	promotion_last_5years	1	0.00	1.00
1	time_spend_company	10	2.00	8.00
2	number_project	7	2.00	5.00
3	left	1	0.00	1.00
4	last_evaluation	1	0.36	0.64
5	average_montly_hours	310	96.00	214.00
6	satisfaction_level	1	0.09	0.91
7	Work_accident	1	0.00	1.00

```
In [55]: # Apply each scaler to the data set to obtain transformed values
         data3 = data2.copy() # create copy of initial data set
         for column in data3.columns:
             if column in min_max_scalers:
                 data3[column] = min_max_scalers[column].transform(pd.DataFrame(data3[column]))

         print("Transformed data set")
         print("="*32)
         data3.head(15)
```

```
Transformed data set
================================
```

```
Out[55]:     satisfaction_level  last_evaluation  number_project  average_montly_hours  \
        0              0.318681         0.265625             0.0              0.285047
        1              0.780220         0.781250             0.6              0.775701
        2              0.021978         0.812500             1.0              0.822430
        3              0.692308         0.796875             0.6              0.593458
        4              0.307692         0.250000             0.0              0.294393
        5              0.351648         0.218750             0.0              0.266355
        6              0.010989         0.640625             0.8              0.705607
        7              0.912088         0.765625             0.6              0.761682
        8              0.879121         1.000000             0.6              0.598131
        9              0.362637         0.265625             0.0              0.214953
        10             0.395604         0.281250             0.0              0.182243
        11             0.021978         0.703125             0.8              0.976636
        12             0.824176         0.875000             0.4              0.644860
        13             0.351648         0.296875             0.0              0.242991
        14             0.296703         0.312500             0.0              0.191589

            time_spend_company  Work_accident  left  promotion_last_5years  sales  \
        0                0.125              0     1                      0      7
        1                0.500              0     1                      0      7
        2                0.250              0     1                      0      7
        3                0.375              0     1                      0      7
        4                0.125              0     1                      0      7
        5                0.125              0     1                      0      7
        6                0.250              0     1                      0      7
        7                0.375              0     1                      0      7
        8                0.375              0     1                      0      7
        9                0.125              0     1                      0      7
        10               0.125              0     1                      0      7
        11               0.250              0     1                      0      7
        12               0.375              0     1                      0      7
        13               0.125              0     1                      0      7
        14               0.125              0     1                      0      7

            salary
        0        1
        1        2
        2        2
        3        1
        4        1
        5        1
        6        1
        7        1
        8        1
```

```
9        1
10       1
11       1
12       1
13       1
14       1
```

In [56]:
```
data3.describe()
```

Out[56]:
```
       satisfaction_level  last_evaluation  number_project  \
count        14999.000000     14999.000000    14999.000000
mean             0.574542         0.556409        0.360611
std              0.273220         0.267452        0.246518
min              0.000000         0.000000        0.000000
25%              0.384615         0.312500        0.200000
50%              0.604396         0.562500        0.400000
75%              0.802198         0.796875        0.600000
max              1.000000         1.000000        1.000000

       average_montly_hours  time_spend_company  Work_accident          left  \
count          14999.000000        14999.000000   14999.000000  14999.000000
mean               0.490889            0.187279       0.144610      0.238083
std                0.233379            0.182517       0.351719      0.425924
min                0.000000            0.000000       0.000000      0.000000
25%                0.280374            0.125000       0.000000      0.000000
50%                0.485981            0.125000       0.000000      0.000000
75%                0.696262            0.250000       0.000000      0.000000
max                1.000000            1.000000       1.000000      1.000000

       promotion_last_5years         sales        salary
count           14999.000000  14999.000000  14999.000000
mean                0.021268      5.870525      1.347290
std                 0.144281      2.868786      0.625819
min                 0.000000      0.000000      0.000000
25%                 0.000000      4.000000      1.000000
50%                 0.000000      7.000000      1.000000
75%                 0.000000      8.000000      2.000000
max                 1.000000      9.000000      2.000000
```

In [57]:
```
# separate our data into dependent (Y) and independent(X) variables
X2_data = data3[['last_evaluation','number_project','average_montly_hours','time_spend_
Y2_data = data3['satisfaction_level']
```

2.1 70/30 Train Test Split

We will split the data using a 70/30 split. i.e. 70% of the data will be randomly chosen to train the model and 30% will be used to evaluate the model

In [58]:
```
from sklearn.model_selection import train_test_split
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_data, Y2_data, test_size=0.3
```

```
In [60]: # Create an instance of linear regression
         reg2 = MLPRegressor()
         reg2.fit(X2_train,y2_train)

Out[60]: MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(100,), learning_rate='constant',
                learning_rate_init=0.001, max_iter=200, momentum=0.9,
                nesterovs_momentum=True, power_t=0.5, random_state=None,
                shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
                verbose=False, warm_start=False)

In [61]: reg2.n_layers_

Out[61]: 3

In [63]: # Make predictions using the testing set
         test2_predicted = reg2.predict(X2_test)
         test2_predicted

Out[63]: array([0.57568627, 0.57893697, 0.5534623 , ..., 0.55752807, 0.71729991,
                0.58278402])

In [64]: # The mean squared error don't worry guys we can do this
         print("Mean squared error: %.2f" % mean_squared_error(y2_test, test2_predicted))

Mean squared error: 0.05
```

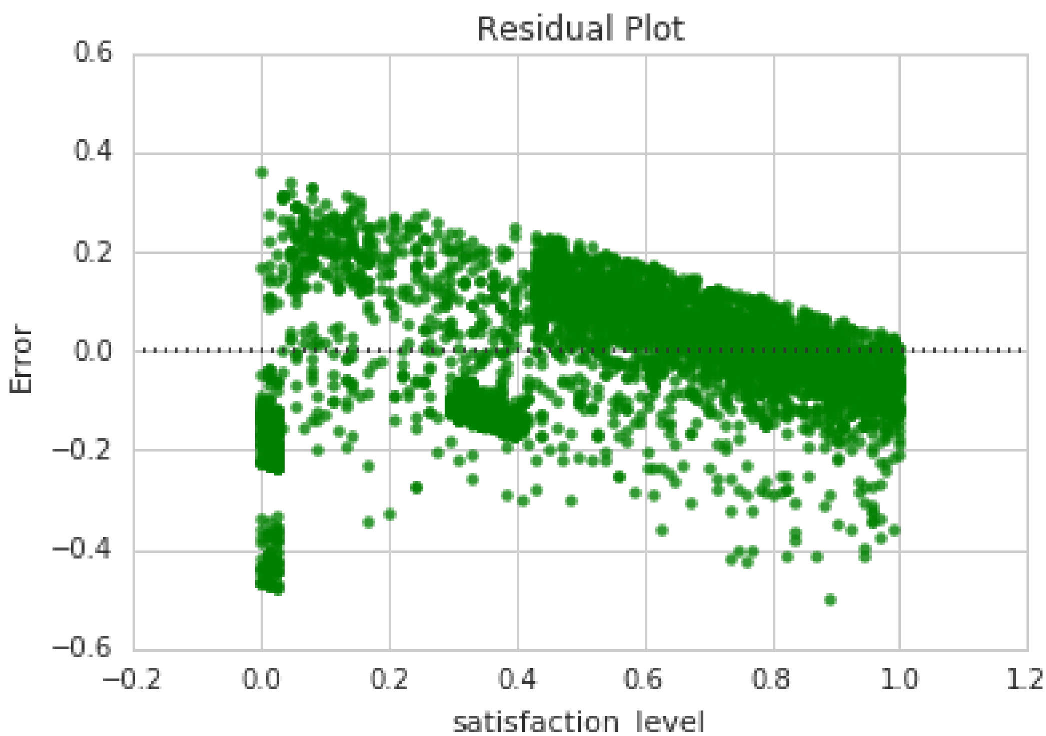
2.2 Hooray, we improved by approximately 0.01

2.3 Let's Visualize using a Residual Plot

```
In [65]: import seaborn as sns
         sns.set(style="whitegrid")

In [71]: sns.residplot(y2_test, test2_predicted,  color="g")
         plt.title("Residual Plot")
         plt.ylabel("Error")

Out[71]: <matplotlib.text.Text at 0x7f1a7da71518>
```



2.4 Did you try changing the amount of hidden layers??