Week 5 - Neural Networks - Classification - 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
https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

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

```
In [39]: data_path = './data/hr_data.csv' # Path to data file
         data = pd.read_csv(data_path)
         data.head(15)
```

```
Out[39]:     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 [40]: # What columns are in the data set ? Do they have spaces that I should consider
         data.columns
```

```
Out[40]: 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 `Department` based on the other factors?

department (column currently named sales) = 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 [41]: 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 [42]: 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_e
```

```
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 [43]:
```python
# 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[43]:

	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 [44]:
```
# separate our data into dependent (Y) and independent(X) variables
X_data = data2[['last_evaluation','number_project','average_montly_hours','time_spend_c
Y_data = data2['sales'] # actually department column
```

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 [45]:
```python
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 [46]:
```python
from sklearn.neural_network import MLPClassifier
```

In [106]:
```python
# Create an instance of linear regression
reg = MLPClassifier()
#reg = MLPClassifier(hidden_layer_sizes=(8,120))
```

In [107]:
```python
reg.fit(X_train,y_train)
```

Out[107]:
```
MLPClassifier(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 [76]:
```python
help(MLPClassifier)
```

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

class MLPClassifier(BaseMultilayerPerceptron, sklearn.base.ClassifierMixin)
 |  Multi-layer Perceptron classifier.
 |
 |  This model optimizes the log-loss function 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
|  ----------
|  classes_ : array or list of array of shape (n_classes,)
|      Class labels for each output.
|
|  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
 |  -----
 |  MLPClassifier 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 numpy arrays or
 |  sparse scipy 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:
 |      MLPClassifier
 |      BaseMultilayerPerceptron
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.base.ClassifierMixin
 |      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.
 |
 |  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.
|
|  predict(self, X)
|      Predict using the multi-layer perceptron classifier
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix}, shape (n_samples, n_features)
|          The input data.
|
|      Returns
|      -------
|      y : array-like, shape (n_samples,) or (n_samples, n_classes)
|          The predicted classes.
|
|  predict_log_proba(self, X)
|      Return the log of probability estimates.
|
|      Parameters
|      ----------
|      X : array-like, shape (n_samples, n_features)
|          The input data.
|
|      Returns
|      -------
|      log_y_prob : array-like, shape (n_samples, n_classes)
|          The predicted log-probability of the sample for each class
|          in the model, where classes are ordered as they are in
|          `self.classes_`. Equivalent to log(predict_proba(X))
|
|  predict_proba(self, X)
|      Probability estimates.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix}, shape (n_samples, n_features)
|          The input data.
|
|      Returns
```

```
 |      -------
 |      y_prob : array-like, shape (n_samples, n_classes)
 |          The predicted probability of the sample for each class in the
 |          model, where classes are ordered as they are in `self.classes_`.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  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.
 |
 |      classes : array, shape (n_classes)
 |          Classes across all calls to partial_fit.
 |          Can be obtained via `np.unique(y_all)`, where y_all is the
 |          target vector of the entire dataset.
 |          This argument is required for the first call to partial_fit
 |          and can be omitted in the subsequent calls.
 |          Note that y doesn't need to contain all labels in `classes`.
 |
 |      Returns
 |      -------
 |      self : returns a trained MLP model.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  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.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Returns the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape = (n_samples, n_features)
 |          Test samples.
 |
```

```
 |      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
 |          True labels for X.
 |
 |      sample_weight : array-like, shape = [n_samples], optional
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of self.predict(X) wrt. y.
```

In [108]:
```
reg.n_layers_ # Number of layers utilized
```

Out[108]:
```
3
```

In [109]:
```
# Make predictions using the testing set
test_predicted = reg.predict(X_test)
test_predicted
```

Out[109]:
```
array([7, 7, 7, ..., 7, 7, 7])
```

In [111]:
```
data3 = X_test.copy()
data3['predicted_department']=test_predicted
data3['predicted_department_en']=label_encoders['sales'].inverse_transform(test_predic
data3['department']=data3['sales']=y_test
data3['department_en']=label_encoders['sales'].inverse_transform(y_test)
data3.head()
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: T
  if diff:
/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: T
  if diff:
```

Out[111]:
```
       last_evaluation  number_project  average_montly_hours  \
7130              0.82               3                   134
13591             0.62               6                   225
14003             0.74               3                   265
3710              0.46               2                   169
12310             0.93               7                   305

       time_spend_company  predicted_department predicted_department_en  \
7130                    3                     7                   sales
13591                   6                     7                   sales
14003                   3                     7                   sales
3710                    2                     7                   sales
12310                   4                     7                   sales
```

```
       department  sales department_en
7130            2      2    accounting
13591           7      7         sales
14003           8      8       support
3710            8      8       support
12310           9      9     technical
```

1.6 Evaluation

1.6.1 Building a Confusion Matrix

NB. Data should be split in training and test data. The model built should be evaluated using unseen or test data

```
In [112]: k=(reg.predict(X_test) == y_test) # Determine how many were predicted correctly
```

```
In [113]: k.value_counts()
```

```
Out[113]: False    3232
          True     1268
          Name: sales, dtype: int64
```

```
In [114]: from sklearn.metrics import confusion_matrix
```

```
In [116]: cm=confusion_matrix(y_test, reg.predict(X_test), labels=y_test.unique())
          cm
```

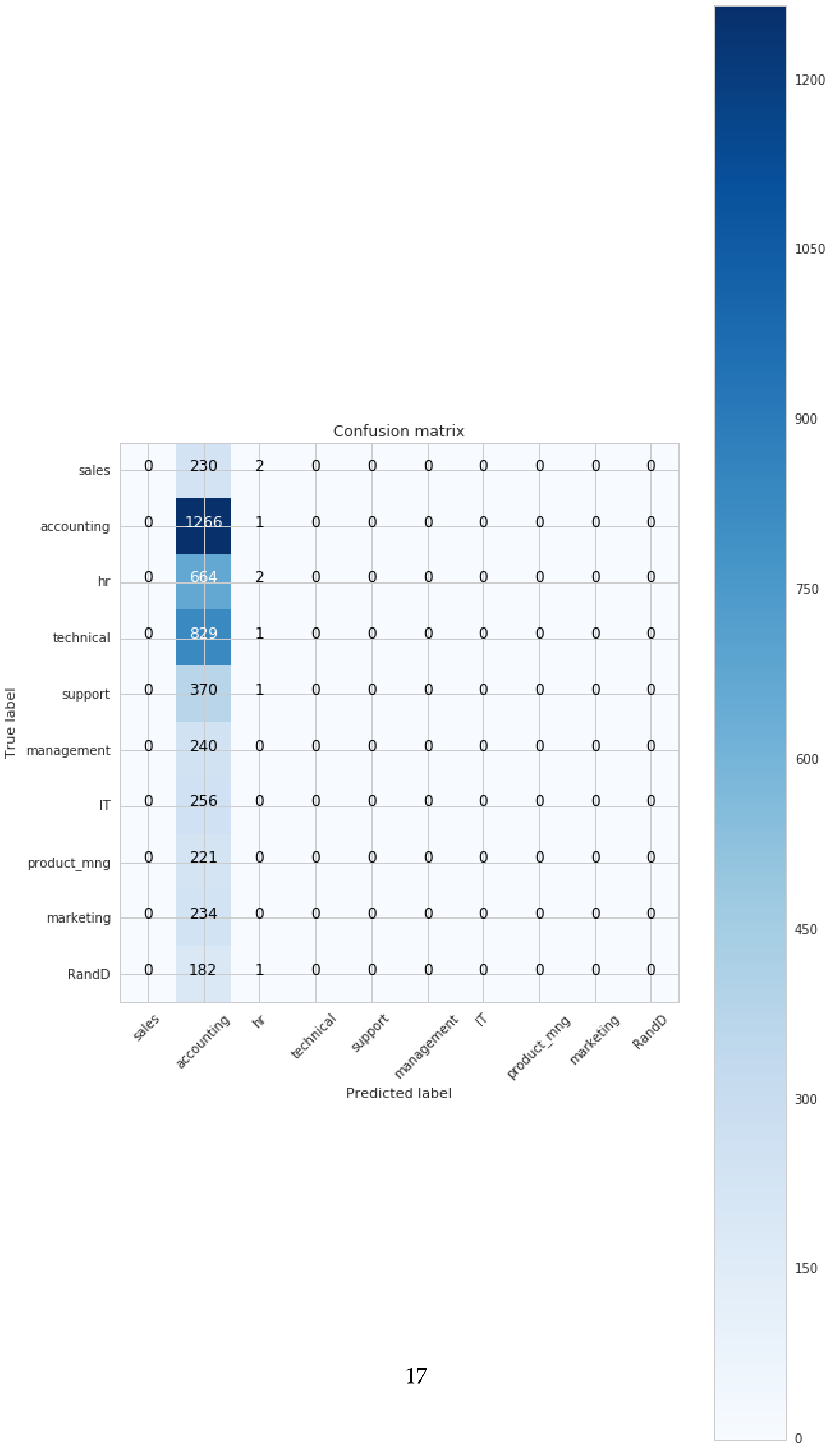
```
Out[116]: array([[   0,  230,    2,    0,    0,    0,    0,    0,    0,    0],
                 [   0, 1266,    1,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  664,    2,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  829,    1,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  370,    1,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  240,    0,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  256,    0,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  221,    0,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  234,    0,    0,    0,    0,    0,    0,    0,    0],
                 [   0,  182,    1,    0,    0,    0,    0,    0,    0,    0]])
```

```
In [117]: def plot_confusion_matrix(cm, classes,
                                    normalize=False,
                                    title='Confusion matrix',
                                    cmap=plt.cm.Blues):
              import itertools
              """
              This function prints and plots the confusion matrix.
              Normalization can be applied by setting `normalize=True`.
              """
              if normalize:
```

```
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
            print("Normalized confusion matrix")
        else:
            print('Confusion matrix, without normalization')

        print(cm)

        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
```

In [122]:
```
plt.figure(figsize=(9,16))
plot_confusion_matrix(cm,data['sales'].unique())
```

```
Confusion matrix, without normalization
[[   0  230    2    0    0    0    0    0    0    0]
 [   0 1266    1    0    0    0    0    0    0    0]
 [   0  664    2    0    0    0    0    0    0    0]
 [   0  829    1    0    0    0    0    0    0    0]
 [   0  370    1    0    0    0    0    0    0    0]
 [   0  240    0    0    0    0    0    0    0    0]
 [   0  256    0    0    0    0    0    0    0    0]
 [   0  221    0    0    0    0    0    0    0    0]
 [   0  234    0    0    0    0    0    0    0    0]
 [   0  182    1    0    0    0    0    0    0    0]]
```


Confusion matrix
True label
Predicted label
sales
accounting
hr
technical
support
management
IT
product_mng
marketing
RandD
0 230 2 0 0 0 0 0 0 0
0 1266 1 0 0 0 0 0 0 0
0 664 2 0 0 0 0 0 0 0
0 829 1 0 0 0 0 0 0 0
0 370 1 0 0 0 0 0 0 0
0 240 0 0 0 0 0 0 0 0
0 256 0 0 0 0 0 0 0 0
0 221 0 0 0 0 0 0 0 0
0 234 0 0 0 0 0 0 0 0
0 182 1 0 0 0 0 0 0 0
1200
1050
900
750
600
450
300
150
0

2 With Preprocessing

In [123]:
```
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 [124]:
```
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[124]:
```
{'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 [125]:
```
#retrieving a scacler
time_spend_company_scaler=min_max_scalers['time_spend_company']
```

In [126]:
```
time_spend_company_scaler
```

Out[126]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [127]:
```
time_spend_company_scaler.data_max_ #Maximum value
```

Out[127]:
```
array([10.])
```

In [128]:
```
time_spend_company_scaler.data_min_ # Minimum value
```

Out[128]:
```
array([2.])
```

In [129]:
```
time_spend_company_scaler.data_range_ # Range = Max- Min
```

Out[129]:
```
array([8.])
```

In [130]:
```python
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[130]:

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

In [131]:
```python
# 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[131]:

	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 [132]:
```
data3.describe()
```

Out[132]:
```
       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 [133]:
```
# 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['sales']
```

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 [134]:
```
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.
```

In [135]:
```
# Create an instance of linear regression
reg2 = MLPClassifier()
reg2.fit(X2_train,y2_train)
```

Out[135]:
```
MLPClassifier(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 [136]:
```
reg2.n_layers_
```

Out[136]:
```
3
```

In [137]:
```
# Make predictions using the testing set
test2_predicted = reg2.predict(X2_test)
test2_predicted
```

Out[137]:
```
array([7, 7, 7, ..., 7, 7, 7])
```

In [138]:
```
data4 = X2_test.copy()
data4['predicted_department']=test2_predicted
data4['predicted_department_en']=label_encoders['sales'].inverse_transform(test2_predi
data4['department']=data3['sales']=y2_test
data4['department_en']=label_encoders['sales'].inverse_transform(y2_test)
data4.head()
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: T
  if diff:
/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:151: DeprecationWarning: T
  if diff:
```

Out[138]:
```
       last_evaluation  number_project  average_montly_hours  \
3058          0.359375             0.2              0.485981
5091          0.218750             0.4              0.602804
12996         0.781250             0.2              0.584112
6296          0.500000             0.6              0.308411
7584          0.968750             0.4              0.808411

       time_spend_company  predicted_department predicted_department_en  \
3058                0.125                     7                   sales
5091                0.125                     7                   sales
12996               0.125                     7                   sales
6296                0.250                     7                   sales
7584                0.125                     7                   sales

       department department_en
3058            9     technical
5091            8       support
12996           7         sales
6296            9     technical
7584            2    accounting
```

2.2 Hooray, we improved by approximately 0.01

2.3 Let's Visualize using a Residual Plot

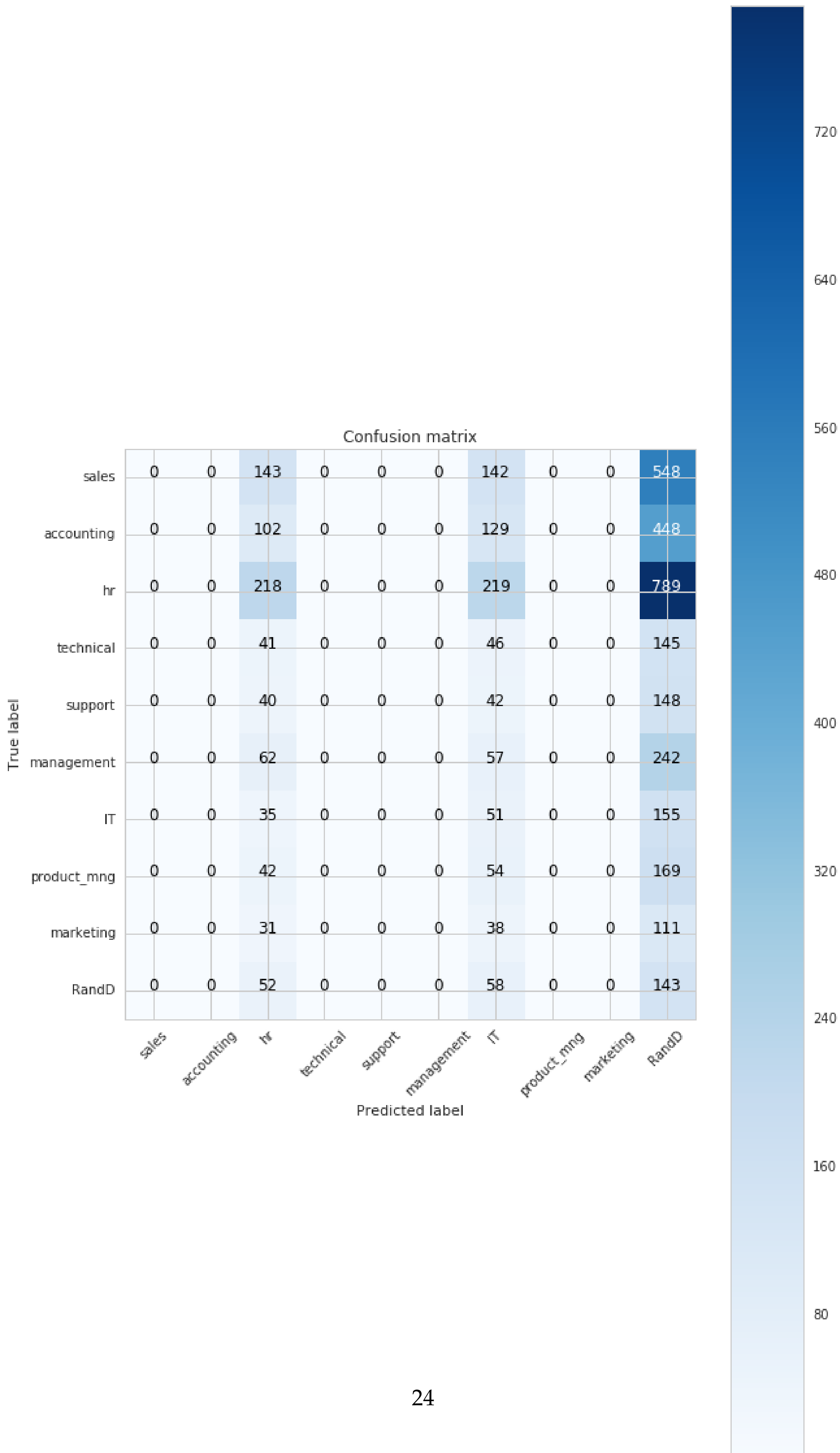
In [142]:
```
k=(reg.predict(X2_test) == y2_test) # Determine how many were predicted correctly
k.value_counts()
```

Out[142]:
```
False    4088
True      412
Name: sales, dtype: int64
```

In [143]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y2_test, reg.predict(X2_test), labels=y2_test.unique())
```

In [144]:
```
plt.figure(figsize=(9,16))
plot_confusion_matrix(cm,data['sales'].unique())
```

```
Confusion matrix, without normalization
[[  0   0 143   0   0   0 142   0   0 548]
 [  0   0 102   0   0   0 129   0   0 448]
 [  0   0 218   0   0   0 219   0   0 789]
 [  0   0  41   0   0   0  46   0   0 145]
 [  0   0  40   0   0   0  42   0   0 148]
 [  0   0  62   0   0   0  57   0   0 242]
 [  0   0  35   0   0   0  51   0   0 155]
 [  0   0  42   0   0   0  54   0   0 169]
 [  0   0  31   0   0   0  38   0   0 111]
 [  0   0  52   0   0   0  58   0   0 143]]
```


Confusion matrix
True label
Predicted label
sales
accounting
hr
technical
support
management
IT
product_mng
marketing
RandD
0 0 143 0 0 0 142 0 0 548
0 0 102 0 0 0 129 0 0 448
0 0 218 0 0 0 219 0 0 789
0 0 41 0 0 0 46 0 0 145
0 0 40 0 0 0 42 0 0 148
0 0 62 0 0 0 57 0 0 242
0 0 35 0 0 0 51 0 0 155
0 0 42 0 0 0 54 0 0 169
0 0 31 0 0 0 38 0 0 111
0 0 52 0 0 0 58 0 0 143
720
640
560
480
400
320
240
160
80
0

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