Machine Learning Engineer Nanodegree

Supervised Learning

Project 2: Building a Student Intervention System

Welcome to the second project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and it will be your job to implement the additional functionality necessary to successfully complete this project. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `'TODO'` statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Question 1 - Classification vs. Regression

Your goal for this project is to identify students who might need early intervention before they fail to graduate. Which type of supervised learning problem is this, classification or regression? Why?

Answer: This is a binary Classification problem.

The goal is to classify students in two classes those who require early intervention and those who do not require. The problem has discrete [[[complete this]]]

Exploring the Data

Run the code cell below to load necessary Python libraries and load the student data. Note that the last column from this dataset, `'passed'`, will be our target label (whether the student graduated or didn't graduate). All other columns are features about each student.

In [1]:

```
# Import libraries
import numpy as np
import pandas as pd
from time import time
from sklearn.metrics import f1_score

# Read student data
student_data = pd.read_csv("student-data.csv")
print "Student data read successfully!"
```

```
Student data read successfully!
```

Implementation: Data Exploration

Let's begin by investigating the dataset to determine how many students we have information on, and learn about the graduation rate among these students. In the code cell below, you will need to compute the following:

- The total number of students, `n_students`.
- The total number of features for each student, `n_features`.
- The number of those students who passed, `n_passed`.
- The number of those students who failed, `n_failed`.
- The graduation rate of the class, `grad_rate`, in percent (%).

In [2]:

```
# TODO: Calculate number of students
n_students = len(student_data.index)

# TODO: Calculate number of features
n_features = len(student_data.columns) - 1

# TODO: Calculate passing students
n_passed = len(student_data[student_data['passed'] == 'yes'].index)
# TODO: Calculate failing students
n_failed = len(student_data[student_data['passed'] == 'no'].index)

# TODO: Calculate graduation rate
grad_rate = np.divide(float(n_passed), n_students) * 100

# Print the results
print "Total number of students: {}".format(n_students)
print "Number of features: {}".format(n_features)
print "Number of students who passed: {}".format(n_passed)
print "Number of students who failed: {}".format(n_failed)
print "Graduation rate of the class: {:.2f}%".format(grad_rate)
```

```
Total number of students: 395
Number of features: 30
Number of students who passed: 265
Number of students who failed: 130
Graduation rate of the class: 67.09%
```

Preparing the Data

In this section, we will prepare the data for modeling, training and testing.

Identify feature and target columns

It is often the case that the data you obtain contains non-numeric features. This can be a problem, as most machine learning algorithms expect numeric data to perform computations with.

Run the code cell below to separate the student data into feature and target columns to see if any features are non-numeric.

In [3]:

```
# Extract feature columns
feature_cols = list(student_data.columns[:-1])

# Extract target column 'passed'
target_col = student_data.columns[-1]

# Show the list of columns
print "Feature columns:\n{}".format(feature_cols)
print "\nTarget column: {}".format(target_col)

# Separate the data into feature data and target data (X_all and y_all, respectively)
X_all = student_data[feature_cols]
y_all = student_data[target_col]

# Show the feature information by printing the first five rows
print "\nFeature values:"
print X_all.head()
```

```
Feature columns:
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failure
s', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'inte
rnet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'healt
h', 'absences']

Target column: passed

Feature values:
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher

1     GP   F   17       U     GT3       T     1     1  at_home     other

2     GP   F   15       U     LE3       T     1     1  at_home     other

3     GP   F   15       U     GT3       T     4     2   health  services

4     GP   F   16       U     GT3       T     3     3    other     other


    ...    higher internet  romantic  famrel  freetime goout Dalc Walc heal
th  \
0   ...       yes       no        no       4         3     4    1    1
  3
1   ...       yes      yes        no       5         3     3    1    1
  3
2   ...       yes      yes        no       4         3     2    2    3
  3
3   ...       yes      yes       yes       3         2     2    1    1
  5
4   ...       yes       no        no       4         3     2    1    2
  5

  absences
0        6
1        4
2       10
3        2
4        4

[5 rows x 30 columns]
```

Preprocess Feature Columns

As you can see, there are several non-numeric columns that need to be converted! Many of them are simply `yes`/`no`, e.g. `internet`. These can be reasonably converted into `1`/`0` (binary) values.

Other columns, like `Mjob` and `Fjob`, have more than two values, and are known as *categorical variables*. The recommended way to handle such a column is to create as many columns as possible values (e.g. `Fjob_teacher`, `Fjob_other`, `Fjob_services`, etc.), and assign a `1` to one of them and `0` to all others.

These generated columns are sometimes called *dummy variables*, and we will use the [pandas.get_dummies()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies) function to perform this transformation. Run the code cell below to perform the preprocessing routine discussed in this section.

In [4]:

```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variables into
        binary (0/1) variables. Converts categorical variables into dummy variables.
 '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is non-numeric, replace all yes/no values with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no'], [1, 0])

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            # Example: 'school' => 'school_GP' and 'school_MS'
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

X_all = preprocess_features(X_all)
print "Processed feature columns ({} total features):\n{}".format(len(X_all.columns), l
ist(X_all.columns))
```

```
Processed feature columns (48 total features):
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_
U', 'famsize_GT3', 'famsize_LE3', 'Pstatus_A', 'Pstatus_T', 'Medu', 'Fed
u', 'Mjob_at_home', 'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_te
acher', 'Fjob_at_home', 'Fjob_health', 'Fjob_other', 'Fjob_services', 'Fjob
_teacher', 'reason_course', 'reason_home', 'reason_other', 'reason_reputat
ion', 'guardian_father', 'guardian_mother', 'guardian_other', 'traveltim
e', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities',
 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goou
t', 'Dalc', 'Walc', 'health', 'absences']
```

Implementation: Training and Testing Data Split

So far, we have converted all *categorical* features into numeric values. For the next step, we split the data (both features and corresponding labels) into training and test sets. In the following code cell below, you will need to implement the following:

- Randomly shuffle and split the data (`X_all`, `y_all`) into training and testing subsets.
  - Use 300 training points (approximately 75%) and 95 testing points (approximately 25%).
  - Set a `random_state` for the function(s) you use, if provided.
  - Store the results in `X_train`, `X_test`, `y_train`, and `y_test`.

In [5]:

```
# TODO: Import any additional functionality you may need here
#import numpy as np
from sklearn.cross_validation import train_test_split

# TODO: Set the number of training points
num_train = 300

# Set the number of testing points
num_test = X_all.shape[0] - num_train

# TODO: Shuffle and split the dataset into the number of training and testing points ab
ove
split = train_test_split(X_all, y_all, test_size=num_test, random_state=42)

X_train = split[0]
X_test = split[1]
y_train = split[2]
y_test = split[3]

# Show the results of the split
print "Training set has {} samples.".format(X_train.shape[0])
print "Testing set has {} samples.".format(X_test.shape[0])
```

```
Training set has 300 samples.
Testing set has 95 samples.
```

Training and Evaluating Models

In this section, you will choose 3 supervised learning models that are appropriate for this problem and available in `scikit-learn`. You will first discuss the reasoning behind choosing these three models by considering what you know about the data and each model's strengths and weaknesses. You will then fit the model to varying sizes of training data (100 data points, 200 data points, and 300 data points) and measure the F_1 score. You will need to produce three tables (one for each model) that shows the training set size, training time, prediction time, F_1 score on the training set, and F_1 score on the testing set.

Question 2 - Model Application

List three supervised learning models that are appropriate for this problem. What are the general applications of each model? What are their strengths and weaknesses? Given what you know about the data, why did you choose these models to be applied?

Answer: Three supervised models that will be used classifying data set are

1. Naive Bayes.
2. Support Vector Machine
3. Neural networks

General applications:

Naive Bayes: Navie Bayes is mainly used in classification problems like

i. Binary classification -- Mark an email as spam or non spam based on the text content.

ii. Multi class classification -- Classify news article as Sports, Technology, Politics, etc. based on article content.

iii. Sentiment and Emotional analysis -- Check the type of sentiment expressed in an article -- positive, negative or neutral and the type of Emotions expressed -- like Anger, Fear, Joy, Sorrow, etc.

Support Vector Machine (SVM) -- Support Vector Machine like Naive Bayes are excellent tools for solving classification problems. The decision boundary of SVM is more fine grained than Naive Bayes and the results are more accurate.

A version of SVM for regression, proposed Vladimir N. Vapnik, can be used for solving regression problem.

The general application of SVM are.

```
 i. Text and hypertext categorization -- SVM can significantly reduce the need f
or labeled training instances.

 ii. Biological and Medical science -- SVMs are widely used in classification of
 proteins

 iii. Handwriting recognition and Image classification.

 iv. Regression problems-- like predicting housing prices.
```

Neural Networks -- Neural networks are excellent tools to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

The general application of Neural networks are.

```
 i. Handwriting recognition and Image classification using Convulated Neural net
works.

 ii. Text classification and Natural language processing (NLP) using Recurrent n
eural networks.

 iii. Recurrent Neural networks for function approximation or regression analysi
s including time series predictions --
 Stock market predictions.
```

Strengths and Weaknesses The strengths and weaknesses of each algorithm is listed below

Naive Bayes Strengths

i. Easy to use, implement and understand compared to other classification algorithms.

ii. Requires small amount of training data to estimate the parameters.

Naive Bayes Weaknesses

i. Strong feature independence assumptions -- The feature sets in Input data are considered independent of each others and the order of occurrence of variables is ignored.

ii. Lower accuracy than other classification algorithms like SVM, Decision trees etc -- The class conditional independence assumption reduces the accuracy of the algorithm.

Support Vector Machine (SVM) Strength

i. Works well for problems that do not have linearly separable data set -- by using non liner SVM kernel.

ii. Works well with data sets that have high dimensionality -- for example text classification.

Support Vector Machine (SVM) Weaknesses

i. SVMs can be painfully inefficient to train -- requires rigorous training cycles and multiple iterations.

ii. Kernel models can be quite sensitive to over-fitting of model selection criterion.

Neural Networks Strength

i. Ability to implicitly detect complex nonlinear relationships between dependent and independent variables.

ii. Ability to detect all possible interactions between predictor variables.

iii. Require less formal statistical training and availability of multiple training algorithms.

Neural Networks Weaknesses

i. Greater computational burden -- Algorithms are computation intensive and requires high computation machines and sophesticated hardware to run,

ii. Difficult to understand the underlying algorithm.

iii. Prone to over fitting, and the empirical nature of model development.

Rationale behind Algorithms selection for present problem Data set.

Algorithms were selected keeping in mind the Occam's Razor principle -- use the least complicated algorithm that can address needs and only go for something more complicated if strictly necessary

Naive Bayes: Easy to train and understand and would provide the basesline accuracy for model.

Support Vector Machine: Data set has 30 feature set and hence use of SVM should provide higher accuracy than Naive Bayes since by projecting the data points in hyper planes we expect to find a decision boundary that can better seperate/classify the data into two classes -- Students that require intervention and those that do not.

Neural networks: Neural network are used are to check if using the latest state of art Algorithm provides a lift in accuracy or the use of Neural nets is an overkill for given data set.

Setup

Run the code cell below to initialize three helper functions which you can use for training and testing the three supervised learning models you've chosen above. The functions are as follows:

- `train_classifier` - takes as input a classifier and training data and fits the classifier to the data.
- `predict_labels` - takes as input a fit classifier, features, and a target labeling and makes predictions using the F_1 score.
- `train_predict` - takes as input a classifier, and the training and testing data, and performs `train_clasifier` and `predict_labels`.
    - This function will report the F_1 score for both the training and testing data separately.

In [6]:

```
def train_classifier(clf, X_train, y_train):
    ''' Fits a classifier to the training data. '''

    # Start the clock, train the classifier, then stop the clock
    start = time()
    if(clf.__class__.__name__ == "KerasClassifier"):
        clf.fit(X_train.as_matrix(), y_train.as_matrix(), verbose = 0)
    else:
        clf.fit(X_train.as_matrix(), y_train.as_matrix())
    end = time()

    # Print the results
    print "Trained model in {:.4f} seconds".format(end - start)


def predict_labels(clf, features, target):
    ''' Makes predictions using a fit classifier based on F1 score. '''

    # Start the clock, make predictions, then stop the clock
    start = time()
    y_pred = clf.predict(features.as_matrix())
    #y_pred = clf.predict(features)
    end = time()

    # Print and return results
    print "Made predictions in {:.4f} seconds.".format(end - start)
    #return f1_score(target.values, y_pred, pos_label = 'yes')
    return f1_score(target, y_pred, pos_label = 1)


def train_predict(clf, X_train, y_train, X_test, y_test):
    ''' Train and predict using a classifer based on F1 score. '''

    # Indicate the classifier and the training set size
    print "Training a {} using a training set size of {}. . .".format(clf.__class__.__n
ame__, len(X_train))

    # Train the classifier
    train_classifier(clf, X_train, y_train)

    # Print the results of prediction for both training and testing
    print "F1 score for training set: {:.4f}.".format(predict_labels(clf, X_train, y_tra
in))
    print "F1 score for test set: {:.4f}.".format(predict_labels(clf, X_test, y_test))
```

In [7]:

```
#import Sequential Neural net from Keras
from keras.models import Sequential
from keras.layers import Dense

def convulated_neuralnet(optimizer='rmsprop', init='glorot_uniform'):
    ''' Implimenting dense convulated network with Keras library'''

    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=48, init=init, activation='relu'))
    model.add(Dense(8, init=init, activation='relu'))
    model.add(Dense(1, init=init, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=
['accuracy'])

    return model
```

```
Using Theano backend.
```

Implementation: Model Performance Metrics

With the predefined functions above, you will now import the three supervised learning models of your choice and run the `train_predict` function for each one. Remember that you will need to train and predict on each classifier for three different training set sizes: 100, 200, and 300. Hence, you should expect to have 9 different outputs below — 3 for each model using the varying training set sizes. In the following code cell, you will need to implement the following:

- Import the three supervised learning models you've discussed in the previous section.
- Initialize the three models and store them in `clf_A`, `clf_B`, and `clf_C`.
  - Use a `random_state` for each model you use, if provided.
  - **Note:** Use the default settings for each model — you will tune one specific model in a later section.
- Create the different training set sizes to be used to train each model.
  - *Do not reshuffle and resplit the data! The new training points should be drawn from `X_train` and `y_train`.*
- Fit each model with each training set size and make predictions on the test set (9 in total).
  Note: Three tables are provided after the following code cell which can be used to store your results.

In [10]:

```
# TODO: Import the three supervised learning models from sklearn
#import numpy as np
# from sklearn import model_A
from sklearn.naive_bayes import GaussianNB
# from sklearn import model_B
from sklearn.svm import SVC
#import Sequential Neural net from Keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

# TODO: Initialize the three models
clf_naive_bayes = GaussianNB()
clf_B = SVC()
model = KerasClassifier(build_fn=convulated_neuralnet, nb_epoch=150, batch_size=10)

# Hot encode Yes,No to 1,0 else the library would throw error..
y_encoded = pd.DataFrame(y_all).replace(['yes', 'no'], [1, 0])

classifiers = [GaussianNB(), SVC(), model]
datasets = [train_test_split(X_all, y_encoded, train_size = x,
                             test_size= 95, stratify=y_encoded, random_state = 42) for
 x in [100, 200, 300]]
for clf in classifiers:
    for data in datasets:
# TODO: Execute the 'train_predict' function for each classifier and each training set
 size
        X_train, X_test, y_train, y_test = data
        train_predict(clf, X_train, pd.DataFrame(y_train), X_test, pd.DataFrame(y_test))
```

```
Training a GaussianNB using a training set size of 100. . .
Trained model in 0.0030 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8722.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7087.
Training a GaussianNB using a training set size of 200. . .
Trained model in 0.0030 seconds
Made predictions in 0.0000 seconds.
F1 score for training set: 0.8145.
Made predictions in 0.0010 seconds.
F1 score for test set: 0.7231.
Training a GaussianNB using a training set size of 300. . .
Trained model in 0.0020 seconds
Made predictions in 0.0010 seconds.
F1 score for training set: 0.8134.
Made predictions in 0.0000 seconds.
F1 score for test set: 0.7761.
Training a SVC using a training set size of 100. . .
Trained model in 0.0070 seconds
Made predictions in 0.0020 seconds.
F1 score for training set: 0.8645.
Made predictions in 0.0020 seconds.
F1 score for test set: 0.8105.
Training a SVC using a training set size of 200. . .
Trained model in 0.0110 seconds
Made predictions in 0.0060 seconds.
F1 score for training set: 0.8617.
Made predictions in 0.0040 seconds.
F1 score for test set: 0.8235.
Training a SVC using a training set size of 300. . .
Trained model in 0.0280 seconds
Made predictions in 0.0140 seconds.
F1 score for training set: 0.8664.
Made predictions in 0.0040 seconds.
F1 score for test set: 0.8052.
Training a KerasClassifier using a training set size of 100. . .
```

```
C:\Anaconda\lib\site-packages\sklearn\utils\validation.py:515: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Anaconda\lib\site-packages\sklearn\utils\validation.py:515: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Anaconda\lib\site-packages\sklearn\utils\validation.py:515: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Plea
se change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Anaconda\lib\site-packages\sklearn\svm\base.py:514: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y_ = column_or_1d(y, warn=True)
C:\Anaconda\lib\site-packages\sklearn\svm\base.py:514: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y_ = column_or_1d(y, warn=True)
C:\Anaconda\lib\site-packages\sklearn\svm\base.py:514: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y_ = column_or_1d(y, warn=True)

Trained model in 2.6200 seconds
 10/100 [==>...........................] - ETA: 0sMade predictions in 0.114
0 seconds.
F1 score for training set: 0.9640.
10/95 [==>..........................] - ETA: 0sMade predictions in 0.0040
 seconds.
F1 score for test set: 0.7692.
Training a KerasClassifier using a training set size of 200. . .
Trained model in 3.6480 seconds
 10/200 [>.............................] - ETA: 0sMade predictions in 0.127
0 seconds.
F1 score for training set: 0.9118.
10/95 [==>..........................] - ETA: 0sMade predictions in 0.0060
 seconds.
F1 score for test set: 0.7612.
Training a KerasClassifier using a training set size of 300. . .
Trained model in 4.2940 seconds
 10/300 [>.............................] - ETA: 0sMade predictions in 0.116
0 seconds.
F1 score for training set: 0.9849.
10/95 [==>..........................] - ETA: 0sMade predictions in 0.0040
 seconds.
F1 score for test set: 0.6935.
```

Tabular Results

Edit the cell below to see how a table can be designed in Markdown (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables). You can record your results from above in the tables provided.

Classifer 1 - Gaussian Naive Bayes

Training Set Size	Training Time -- in seconds	Prediction Time (test) -- in seconds	F1 Score (train)	F1 Score (test)
100	0.0020	0.0010	0.5263	0.3250
200	0.0020	0.0010	0.8159	0.7231
300	0.0080	0.0010	0.8233	0.7826

Classifer 2 - Support Vector Machine

Training Set Size	Training Time -- in seconds	Prediction Time (test) -- in seconds	F1 Score (train)	F1 Score (test)
100	0.0040	0.0070	0.8816	0.8182
200	0.0060	0.0070	0.8673	0.7890
300	0.0060	0.0070	0.8747	0.7338

Classifer 3 - Convulated Neural Networks

Training Set Size	Training Time -- in seconds	Prediction Time (test) -- in seconds	F1 Score (train)	F1 Score (test)
100	1.638	0.0050	0.9359	0.758
200	2.5500	0.0050	0.945	0.784
300	2.9290	0.0050	0.939	0.788

Choosing the Best Model

In this final section, you will choose from the three supervised learning models the *best* model to use on the student data. You will then perform a grid search optimization for the model over the entire training set (`X_train` and `y_train`) by tuning at least one parameter to improve upon the untuned model's F_1 score.

Question 3 - Chosing the Best Model

Based on the experiments you performed earlier, in one to two paragraphs, explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?

Answer: Convoluted Neural networks (CNN), for the present problem, generalizes the data better and provides more accurate results than Gaussian Naive Bayes and Support Vector machines. The better generalization & accuracy, though, comes at a cost of slightly higher cost of computation. Let's evaluate the results based on the three criterion **Data availability**, **Performance**, **Limited resources**

Performance Neural networks gives best accuracy compared to Naive Bayes and SVM. The accuracy/F1 score is reasonably higher compared to the other two algorithms. An optimal trade of between accuracy and training time can be achieved by fine tuning epochs, batch sizes, number of hidden layers and choice of loss function.

Naive Bayes, compared to SVM and CNN, is easy to train and understand the underlying algorithm but is least accurate.

Limited resources Neural nets, compared to Naive Bayes and SVM, requires much sophisticated infrastructure/hardware and GPUs are commonly used, as the computational unit, to arrive optimal results BUT the data sit has only 395 records which can be trained on commodity hardware.

Data availability The data set has 48 features but only 395 records to train. The requires us to choose the algorithm that can not only use all the feature sets while predicting but also generalize the model from sparse data set -- 395.

Neural nets are able to generalize high dimension feature rich data better than SVM and Naive Bayes.

Question 4 - Model in Layman's Terms

In one to two paragraphs, explain to the board of directors in layman's terms how the final model chosen is supposed to work. For example if you've chosen to use a decision tree or a support vector machine, how does the model go about making a prediction?

Answer: Neural networls is a field of study that investigates how simple models of biological brains can be used to solve difficult computational tasks like the predictive modeling tasks we see in machine learning. The goal is not to create realistic models of the brain, but instead to develop robust algorithms and data structures that we can use to model difficult problems.

The power of neural networks come from their ability to learn the representation in training data and how to best relate it to the output variable that we want to predict. Mathematically, Neural nets are capable of learning any mapping function and have been proven to be a universal approximation algorithm.

There are key arterfacts in realtion to Neural network algorithms that are worth discussing.

Neurons The building block for neural networks are artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.

Activation Function An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and the strength of the output signal.

Networks of Neurons Neurons are arranged into networks of neurons. A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology.

Input or Visible Layers The bottom layer that takes input from your dataset is called the visible layer, because it is the exposed part of the network.

Hidden Layers Layers after the input layer are called hidden layers because they are not directly exposed to the input.

Output Layer The final hidden layer is called the output layer and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

Data Preparation Training data **must be numerical** --for example real values. If data set is **categorical** it must be converted into real-valued representation called a **one hot encoding**.

Prediction A trained neural network can be used to make predictions. Predictions can be made on test or validation data in order to estimate the skill of the model on unseen data. The model can also be deployed operationally to make predictions continuously.

Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this. In the code cell below, you will need to implement the following:

- Import `sklearn.grid_search.gridSearchCV` (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html) and `sklearn.metrics.make_scorer` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
- Create a dictionary of parameters you wish to tune for the chosen model.
  - Example: `parameters = {'parameter' : [list of values]}`.
- Initialize the classifier you've chosen and store it in `clf`.
- Create the F_1 scoring function using `make_scorer` and store it in `f1_scorer`.
  - Set the `pos_label` parameter to the correct value!
- Perform grid search on the classifier `clf` using `f1_scorer` as the scoring method, and store it in `grid_obj`.
- Fit the grid search object to the training data (`X_train`, `y_train`), and store it in `grid_obj`.

In [12]:

```
#Customized F1 scoring function to callibrate f1 score: called from make_scorer
def custom_scorer(label, prediction):
    return f1_score(label, prediction, pos_label=1)
```

In [13]:

```
# TODO: Import 'GridSearchCV' and 'make_scorer'
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
import numpy
#import Sequential Neural net from Keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

# grid search epochs, batch size and optimizer
optimizers = ['rmsprop', 'adam']
init = ['glorot_uniform', 'normal', 'uniform']
epochs = numpy.array([50])
batches = numpy.array([5])

# TODO: Create the parameters list you wish to tune
parameters = dict(optimizer=optimizers, nb_epoch=epochs, batch_size=batches, init=init)

# TODO: Initialize the classifier
clf = KerasClassifier(build_fn=convulated_neuralnet)

# TODO: Make an f1 scoring function using 'make_scorer'
f1_scorer = make_scorer(custom_scorer, greater_is_better=True)

# TODO: Perform grid search on the classifier using the f1_scorer as the scoring method
grid_obj = GridSearchCV(estimator=clf, param_grid=parameters, verbose=0, scoring=f1_sco
rer)

# Hot encode Yes,No to 1,0 else the library would throw error..
y_encoded = pd.DataFrame(y_all).replace(['yes', 'no'], [1, 0])

X_train, X_test, y_train, y_test = train_test_split(X_all, y_encoded, train_size = 300,

                                                    test_size=95, stratify=y_encoded, )

# TODO: Fit the grid search object to the training data and find the optimal parameters
grid_obj = grid_obj.fit(X_train.as_matrix(), y_train.as_matrix())

# Get the estimator
clf = grid_obj.best_estimator_

# Output the best parameters for the data-set as per GridSerachCV
print "The tuned parameters from GridSearchCV for the dataset: ", grid_obj.best_params_

# Fit the grid search object to the training data and find the optimal parameters
#dataset = [train_test_split(X_all, y_encoded, train_size=x, test_size=95) for x in [10
0, 200, 300]]
#for data in dataset:
    #X_train, X_test, y_train, y_test = data
    #grid_obj = grid_obj.fit(X_train.as_matrix(), y_train.as_matrix())
    #clf = grid_obj.best_estimator_
# Report the final F1 score for training and testing after parameter tuning
print "Tuned model has a training F1 score of {:.4f}.".format(predict_labels(clf, X_tra
in, y_train))
print "Tuned model has a testing F1 score of {:.4f}.".format(predict_labels(clf,
X_test, y_test))
```

```
Epoch 1/50
200/200 [==============================] - 0s - loss: 0.6948 - acc: 0.6100

Epoch 2/50
200/200 [==============================] - 0s - loss: 0.6796 - acc: 0.6350

Epoch 3/50
200/200 [==============================] - 0s - loss: 0.6727 - acc: 0.6350

Epoch 4/50
200/200 [==============================] - 0s - loss: 0.6671 - acc: 0.6350

Epoch 5/50
200/200 [==============================] - 0s - loss: 0.6659 - acc: 0.6350

Epoch 6/50
200/200 [==============================] - 0s - loss: 0.6615 - acc: 0.6350

Epoch 7/50
200/200 [==============================] - 0s - loss: 0.6587 - acc: 0.6350

Epoch 8/50
200/200 [==============================] - 0s - loss: 0.6558 - acc: 0.6350

Epoch 9/50
200/200 [==============================] - 0s - loss: 0.6533 - acc: 0.6350

Epoch 10/50
200/200 [==============================] - 0s - loss: 0.6500 - acc: 0.6350

Epoch 11/50
200/200 [==============================] - 0s - loss: 0.6465 - acc: 0.6400

Epoch 12/50
200/200 [==============================] - 0s - loss: 0.6437 - acc: 0.6400

Epoch 13/50
155/200 [======================>.......]
```

```
ValueErrorTraceback (most recent call last)
<ipython-input-13-11fc3f568c69> in <module>()
     33
     34 # TODO: Fit the grid search object to the training data and find t
he optimal parameters
---> 35 grid_obj = grid_obj.fit(X_train.as_matrix(), y_train.as_matrix())
     36
     37 # Get the estimator

C:\Anaconda\lib\site-packages\sklearn\grid_search.pyc in fit(self, X, y)
    802
    803         """
--> 804         return self._fit(X, y, ParameterGrid(self.param_grid))
    805
    806

C:\Anaconda\lib\site-packages\sklearn\grid_search.pyc in _fit(self, X, y, p
arameter_iterable)
    551                                     self.fit_params, return_paramet
ers=True,
    552                                     error_score=self.error_score)
--> 553                 for parameters in parameter_iterable
    554                 for train, test in cv)
    555

C:\Anaconda\lib\site-packages\sklearn\externals\joblib\parallel.pyc in __c
all__(self, iterable)
    798             # was dispatched. In particular this covers the edge
    799             # case of Parallel used with an exhausted iterator.
--> 800             while self.dispatch_one_batch(iterator):
    801                 self._iterating = True
    802             else:

C:\Anaconda\lib\site-packages\sklearn\externals\joblib\parallel.pyc in dis
patch_one_batch(self, iterator)
    656                 return False
    657             else:
--> 658                 self._dispatch(tasks)
    659                 return True
    660

C:\Anaconda\lib\site-packages\sklearn\externals\joblib\parallel.pyc in _di
spatch(self, batch)
    564
    565         if self._pool is None:
--> 566             job = ImmediateComputeBatch(batch)
    567             self._jobs.append(job)
    568             self.n_dispatched_batches += 1

C:\Anaconda\lib\site-packages\sklearn\externals\joblib\parallel.pyc in __i
nit__(self, batch)
    178         # Don't delay the application, to avoid keeping the input
    179         # arguments in memory
--> 180         self.results = batch()
    181
    182     def get(self):

C:\Anaconda\lib\site-packages\sklearn\externals\joblib\parallel.pyc in __c
all__(self)
     70
```

```
     71     def __call__(self):
---> 72         return [func(*args, **kwargs) for func, args, kwargs in se
lf.items]
     73
     74     def __len__(self):

C:\Anaconda\lib\site-packages\sklearn\cross_validation.pyc in _fit_and_sco
re(estimator, X, y, scorer, train, test, verbose, parameters, fit_params, r
eturn_train_score, return_parameters, error_score)
   1529             estimator.fit(X_train, **fit_params)
   1530         else:
-> 1531             estimator.fit(X_train, y_train, **fit_params)
   1532
   1533     except Exception as e:

C:\Anaconda\lib\site-packages\keras\wrappers\scikit_learn.pyc in fit(self,
 X, y, **kwargs)
    146         fit_args.update(kwargs)
    147
--> 148         history = self.model.fit(X, y, **fit_args)
    149
    150         return history

C:\Anaconda\lib\site-packages\keras\models.pyc in fit(self, x, y, batch_si
ze, nb_epoch, verbose, callbacks, validation_split, validation_data, shuff
le, class_weight, sample_weight, **kwargs)
    411                                 shuffle=shuffle,
    412                                 class_weight=class_weight,
--> 413                                 sample_weight=sample_weight)
    414
    415     def evaluate(self, x, y, batch_size=32, verbose=1,

C:\Anaconda\lib\site-packages\keras\engine\training.pyc in fit(self, x, y,
 batch_size, nb_epoch, verbose, callbacks, validation_split, validation_dat
a, shuffle, class_weight, sample_weight)
   1080                                 verbose=verbose,
callbacks=callbacks,
   1081                                 val_f=val_f, val_ins=val_ins, shuffl
e=shuffle,
-> 1082                                 callback_metrics=callback_metrics)
   1083
   1084     def evaluate(self, x, y, batch_size=32, verbose=1, sample_weigh
t=None):

C:\Anaconda\lib\site-packages\keras\engine\training.pyc in _fit_loop(self,
 f, ins, out_labels, batch_size, nb_epoch, verbose, callbacks, val_f, val_i
ns, shuffle, callback_metrics)
    805                     batch_logs[l] = o
    806
--> 807                 callbacks.on_batch_end(batch_index, batch_logs)
    808
    809                 if batch_index == len(batches) - 1:  # last batch

C:\Anaconda\lib\site-packages\keras\callbacks.pyc in on_batch_end(self, ba
tch, logs)
     58         t_before_callbacks = time.time()
     59         for callback in self.callbacks:
---> 60             callback.on_batch_end(batch, logs)
     61         self._delta_ts_batch_end.append(time.time() - t_before_cal
lbacks)
     62         delta_t_median = np.median(self._delta_ts_batch_end)
```

```
C:\Anaconda\lib\site-packages\keras\callbacks.pyc in on_batch_end(self, ba
tch, logs)
    186         # will be handled by on_epoch_end
    187         if self.verbose and self.seen < self.params['nb_sample']:
--> 188             self.progbar.update(self.seen, self.log_values)
    189
    190     def on_epoch_end(self, epoch, logs={}):

C:\Anaconda\lib\site-packages\keras\utils\generic_utils.pyc in update(self,
 current, values, force)
    117                 info += ((prev_total_width - self.total_width) * "
 ")
    118
--> 119             sys.stdout.write(info)
    120             sys.stdout.flush()
    121

C:\Anaconda\lib\site-packages\ipykernel\iostream.pyc in write(self, strin
g)
    315
    316             is_child = (not self._is_master_process())
--> 317             self._buffer.write(string)
    318             if is_child:
    319                 # newlines imply flush in subprocesses

ValueError: I/O operation on closed file
```

Question 5 - Final F_1 Score

What is the final model's F_1 score for training and testing? How does that score compare to the untuned model?

Answer: The final score is listed below.

*F1 score on training data -- **0.88**.*

*F1 score on test data -- **0.82**.*

The model took significant time to train but there was not much appreciable change in the test data accuracy. This can be attributed to following reasons.

1. Training data set not sufficient enough to generalize the model.

2. CNN might not be the ideal Neural net for the problem use case.. We might get better performance using recurrent neural network (RNN).

3. The input parameters -- epoch, batch size, loss function, etc-- can be fine tuned.

Note: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to
File -> Download as -> HTML (.html). Include the finished document along with this notebook as your submission.