PHYS243: Foundation of Applied Machine Learning

Homework 5 - Find Income with SVM
Prof. Bahram Mobasher, Inst. Abtin Shahidi
Submitted By:
Ray Felipe
Student ID: 862120029
Aug. 14, 2019

Executive Summary

Random Forest and SVM classification are two important machine learning algorithms used for predicting data clases. In this exercise, we'll use these algorithms to predict a given salary of an individual. We'll use data features to predict whether an employee earns more or less and 50,000 per year. These features can either be education level and class of employment.

We'll also apply accuracy measures to determine the accuracy of our prediction based on random forest or SVM algorithms.

1.0 Find the income using Support Vector Machines! From the link to adult.zip download the data set.

The data for this project is from the census bureau database found at [http://www.census.gov/ftp/pub/DES/www/welcome.html (http://www.census.gov/ftp/pub/DES/www/welcome.html)](http://www.census.gov/ftp/pub/DES/www/welcome.html). It contains a set of features for a given individual along with salary.

1.1 First, take a look at the data. You can see that the data contains categorical data as well.

It is always important to get a good overview understanding of the data. Understanding its structures and layout is necessary before any processing and analysis is applied.

In [1]:
```
import pandas as pd
df = pd.read_csv('dataset/adult.data', header=None)
df.shape
```

Out[1]: (32561, 15)

The shape tells us that there are 15 data features along with 33,000 data instances.

Let's add the column names as outlined in the included definition in the downloaded data set. Adding column names will allow us to better process our data and apply the necessary algorithms.

In [2]:
```
df.columns=['age','workclass','fnlwgt','education','education-num','marital-status
','occupation','relationship','race','sex','capital-gain','capital-loss','hours-per
-week','native-country','salary']
df.head()
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0

1.2 Run a random forests and measure your performance.

Leveraging the code from Instructor Abtin's lecture notes, I ran the data through the random forest function.

In [3]:

```
import numpy as np
import random
import matplotlib.pyplot as plt

def Entropy(prob_X):
    import math
    _sum_ = 0
    _tot_ = 0
    # checks
    for prob in prob_X:
        assert prob >= 0, "Negative probability is not accepted!!!"
        _tot_ += prob
    for prob in prob_X:
        if _tot_==0:
            continue

        prob = prob/_tot_
        if prob == 0:
            pass
        else:
            _sum_ += prob * math.log2(prob)
    return abs(_sum_)


def Boolean_Entropy(q):
    assert q >= 0 and q <= 1, "q = {} is not between [0,1]!".format(q)
    return Entropy([q, 1-q])

def Boolean_Entropy_counts(p, n):
    if n==0 and p==0:
        return 0
    q = p / (n + p)
    return Boolean_Entropy(q)
```

In [4]:
```
def Remainder_Entropy(Attr, outcome):
    set_of_distinct_values = set(Attr)
    count_distict_values = len(set_of_distinct_values)
    count_distict_outcomes = len(set(outcome))

    assert count_distict_outcomes <= 2, "{} different outcomes but expected Boolea
n"

    count_total_positives = len([i for i in outcome if i!=0])
    count_total_negatives = len(outcome) - count_total_positives

    import numpy as np

    Attr_np = np.array(Attr)
    outcome_np = np.array(outcome)

    _sum_ = 0

    for value in set_of_distinct_values:
        _outcome_ = outcome_np[Attr_np==value]
        count_positives = len([i for i in _outcome_ if i!=0])
        count_negatives = len(_outcome_) - count_positives
        _entropy_ = Boolean_Entropy_counts(count_positives, count_negatives)
        _weights_ = (count_positives + count_negatives)
        _weights_ = _weights_ / (count_total_positives + count_total_negatives)
        _sum_ += _weights_ * _entropy_
    return _sum_

def Information_Gain(Attr, outcome):
    count_total_positives = len([i for i in outcome if i!=0])
    count_total_negatives = len(outcome) - count_total_positives
    initial_entropy = Boolean_Entropy_counts(count_total_positives, count_total_neg
atives)
    remaining_entropy = Remainder_Entropy(Attr, outcome)
    info_gain = initial_entropy - remaining_entropy
    return info_gain
```

In [5]:
```
import copy
import math
import random

from statistics import mean, stdev
from collections import defaultdict

def euclidean_distance(X, Y):
    return math.sqrt(sum((x - y)**2 for x, y in zip(X, Y)))

def cross_entropy_loss(X, Y):
    n=len(X)
    return (-1.0/n)*sum(x*math.log(y) + (1-x)*math.log(1-y) for x, y in zip(X, Y))

def rms_error(X, Y):
    return math.sqrt(ms_error(X, Y))

def ms_error(X, Y):
    return mean((x - y)**2 for x, y in zip(X, Y))

def mean_error(X, Y):
    return mean(abs(x - y) for x, y in zip(X, Y))

def manhattan_distance(X, Y):
    return sum(abs(x - y) for x, y in zip(X, Y))

def mean_boolean_error(X, Y):
    return mean(int(x != y) for x, y in zip(X, Y))

def hamming_distance(X, Y):
    return sum(x != y for x, y in zip(X, Y))

def _read_data_set(data_file, skiprows=0, separator=None):
    with open(data_file, "r") as f:
        file = f.read()
        lines = file.splitlines()
        lines = lines[skiprows:]

    data_ = [[] for _ in range(len(lines))]

    for i, line in enumerate(lines):
        splitted_line = line.split(separator)
        float_line = []
        for value in splitted_line:
            try:
                value = float(value)
            except ValueError:
                if value=="":
                    continue
                else:
                    pass
            float_line.append(value)
        if float_line:
            data_[i] = float_line

    for line in data_:
        if not line:
            data_.remove(line)

    return data_

def unique(seq):
    return list(set(seq))
```

In [6]:
```
class Data_Set:
    def __init__(self, examples=None, attributes=None,  attribute_names=None,
                 target_attribute = -1,  input_attributes=None,  values=None,
                 distance_measure = mean_boolean_error, name='',  source='',
                 excluded_attributes=(), file_info=None):

        self.file_info = file_info
        self.name = name
        self.source = source
        self.values = values
        self.distance = distance_measure
        self.check_values_flag = bool(values)

        # Initialize examples from a list
        if examples is not None:
            self.examples = examples
        elif file_info is None:
            raise ValueError("No Examples! and No Address!")
        else:
            self.examples = _read_data_set(file_info[0], file_info[1], file_info
[2])

        # Attributes are the index of examples. can be overwrite
        if self.examples is not None and attributes is None:
            attributes = list(range(len(self.examples[0])))

        self.attributes = attributes

        # Initialize attribute_names from string, list, or to default
        if isinstance(attribute_names, str):
            self.attribute_names = attribute_names.split()
        else:
            self.attribute_names = attribute_names or attributes

        # set the definitions needed for the problem
        self.set_problem(target_attribute, input_attributes=input_attributes,
                         excluded_attributes=excluded_attributes)

    def get_attribute_num(self, attribute):
        if isinstance(attribute, str):
            return self.attribute_names.index(attribute)
        else:
            return attribute

    def set_problem(self, target_attribute, input_attributes=None, excluded_attribu
tes=()):

        self.target_attribute = self.get_attribute_num(target_attribute)

        exclude = [self.get_attribute_num(excluded) for excluded in excluded_attrib
utes]

        if input_attributes:
            self.input_attributes = remove_all(self.target_attribute, input_attribu
tes)
        else:
            inputs = []
            for a in self.attributes:
                if a != self.target_attribute and a not in exclude:
                    inputs.append(a)
            self.input_attributes = inputs

        if not self.values:
            self.update_values()
```

In [7]:
```
class Decision_Branch:

    def __init__(self, attribute, attribute_name=None, default_child=None, branche
s=None):
        """Initialize by specifying what attribute this node tests."""

        self.attribute = attribute
        self.attribute_name = attribute_name or attribute
        self.default_child = default_child
        self.branches = branches or {}

    def __call__(self, example):
        """Classify a given example using the attribute and the branches."""
        attribute_val = example[self.attribute]
        if attribute_val in self.branches:
            return self.branches[attribute_val](example)
        else:
            # return default class when attribute is unknown
            return self.default_child(example)

    def add(self, value, subtree):
        """Add a branch.  If self.attribute = value, move to the given subtree."""
        self.branches[value] = subtree

    def display_out(self, indent=0):
        name = self.attribute_name
        print("Test", name)
        for value, subtree in self.branches.items():
            print(" " * indent * 5, name, '=', value, "--->", end=" ")
            subtree.display_out(indent + 1)
        # New line
        print()

    def __repr__(self):
        return ('Decision_Branch({}, {}, {})'
                .format(self.attribute, self.attribute_name, self.branches))

class Decision_Leaf:
    """A simple leaf class for a decision tree that hold a result."""

    def __init__(self, result):
        self.result = result

    def __call__(self, example):
        return self.result

    def display_out(self, indent=0):
        print('RESULT =', self.result)

    def __repr__(self):
        return repr(self.result)

def Decision_Tree_Learner(dataset):
    """
    Learning Algorithm for a Decision Tree
    """

    target, values = dataset.target_attribute, dataset.values

    def decision_tree_learning(examples, attrs, parent_examples=()):
        if not examples:
            return plurality(parent_examples)
        elif same_class_for_all(examples):
            return Decision_Leaf(examples[0][target])
```

In [8]:
```
def Random_Forest(dataset, n=5, verbose=False):
    def data_bagging(dataset, m=0):
        """Sample m examples with replacement"""
        n = len(dataset.examples)
        return weighted_sample_with_replacement(m or n, dataset.examples, [1]*n)

    def feature_bagging(dataset, p=0.7):
        """Feature bagging with probability p to retain an attribute"""
        inputs = [i for i in dataset.input_attributes if probability(p)]
        return inputs or dataset.input_attributes

    def predict(example):
        if verbose:
            print([predictor(example) for predictor in predictors])
        return mode(predictor(example) for predictor in predictors)

    predictors = [Decision_Tree_Learner(Data_Set(examples=data_bagging(dataset),
                                                 attributes=dataset.attributes,
                                                 attribute_names=dataset.attribute_
names,
                                                 target_attribute=dataset.target_at
tribute,
                                                 input_attributes=feature_bagging(d
ataset))) for _ in range(n)]

    return predict
```

Now let's load the adult.data data set.

In [9]:
```
address = "dataset/adult.data"
attr_names = ['age','fnlwgt','workclass','education','education-num','marital-statu
s','occupation','relationship','race','sex','capital-gain','capital-loss','hours-pe
r-week','native-country','salary']
Adult_Data = Data_Set(name = "Adults", target_attribute=4,
                      file_info=(address, 0, ","), attribute_names=attr_names)
```

Next, we split the data into training and test set using the *train_test_split()* function so that we can perform accuracy measurement of our random forest prediction.

In [10]:
```
Adult_Data_train, Adult_Data_test = Adult_Data.train_test_split(test_fraction=0.2,
Seed=90)
RF_Adult_Data = Random_Forest(Adult_Data_train, n = 2)
```

In [11]:
```
def Measure_accuracy(true_values, predictions):
    _sum_ = 0
    for truth, prediction in zip(true_values, predictions):
        if truth==prediction:
            _sum_+=1
    return _sum_/len(predictions)

rf_predictions = [RF_Adult_Data(a[:-1]) for a in Adult_Data_test.examples]
true_Test_values = [example[Adult_Data_test.target_attribute] for example in Adult_
Data_test.examples]
measure_accuracy_output = Measure_accuracy(true_Test_values, rf_predictions)
measure_accuracy_output
```

Out[11]:
```
1.0
```

The outcome of our accuracy measurement above indicates that our random forest prediction is 1.0 accurate.

2.0 Transform Categorical data into numbers!

2.1 Now that we have more complicated algorithm, let's make use out of them. But first you should change your categorical data into real valued number which are needed for the SVM algorithm. Come up with a method that can do this translation.

There are several methods for transforming categorical data into real valued numbers. These methods are label encoding, one hot encoding, and binary encoding among others. In this exercise I used one hot encoding.

One hot encoding converts each category value into a new column and assigns a 1 or 0 (True/False) value to the column. This has the benefit of not weighting a value improperly but does have the downside of adding more columns to the data set[1]. Given that our data set does not have many variations in a feature data value, one hot encoding will not have this downside with respect to our dataset.

To apply this transformation, I leverage Pandas one-hot encoding library called get_dummies function.

3.0 Scale your attributes!

3.1 Now that you have numerical attributes, scale all your features to something reasonable.

Let's now perform attribute scaling using one hot encoding

In [12]:
```
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

df = pd.read_csv('dataset/adult.data', header=None)
df.columns=['age','workclass','fnlwgt','education','education-num','marital-status
','occupation','relationship','race','sex','capital-gain','capital-loss','hours-per
-week','native-country','salary']
obj_df = df.select_dtypes(include=['object']).copy()

# One hot encoding
pd_encoded_output = pd.get_dummies(obj_df, columns=["workclass","education","marita
l-status","occupation","relationship","race","sex","native-country","salary"])
pd_encoded_output.head()
```

Out[12]:

	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	workclass_ Private	workclass_ Self-emp-inc	workclass_ Self-emp-not-inc	workclass_ State-gov	workclas Witho p
0	0	0	0	0	0	0	0	1	
1	0	0	0	0	0	0	1	0	
2	0	0	0	0	1	0	0	0	
3	0	0	0	0	1	0	0	0	
4	0	0	0	0	1	0	0	0	

As shown in the output above, get_dummies created additional columns prefixed by their original column names. The values for these features have been replace by number 0 or 1.

4.0 Run Support Vector Machine!

4.1 Now that you preprocessed your data, you can run the algorithm and measure your performance.

Before we can run our SVM algorithm, we must preprocess the data.

In [13]:
```
from sklearn import svm

## Now let's classify. Replace X, y above by building our paired feature
X = []
y = [] # this will hold the true values for prediction
for i in range(len(pd_encoded_output["workclass_ Federal-gov"])):
    feature_pair = []
    feature_for_prediction = []
    feature_pair.append(pd_encoded_output["workclass_ Federal-gov"][i])
    feature_pair.append(pd_encoded_output["education_ Bachelors"][i])
    feature_for_prediction.append(pd_encoded_output["salary_ >50K"][i])
    X.append(feature_pair)
    y.append(feature_for_prediction)
```

In the above code, I created an X and y variable. The X will contain the selected feature pair that we need to predict. In this case, I used "workclass" and "education". The y will contain the predicted feature, in this case, "salary".

Let's fit these data into our SVM model.

In [14]:
```
clf = svm.SVC(gamma='scale')
#print("clf.fit(X, y):")
clf.fit(X, y)
```

```
C:\Users\ramon\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761: Data
ConversionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[14]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Now that our data has been fitted, let's predict the salary for our **workclass** and **education** feature pair.

In [18]:
```
feature_pair_for_measure_accuracy = []
feature_pair_svm_predicted_values = []
for i in range(len(pd_encoded_output["workclass_ Federal-gov"])):
    feature_pair = []
    feature_for_prediction = []
    feature_pair.append(pd_encoded_output["workclass_ Federal-gov"][i])
    feature_pair.append(pd_encoded_output["education_ Bachelors"][i])
    #feature_for_prediction.append(pd_encoded_output["salary_ >50K"][i])
    feature_pair_for_measure_accuracy.append(feature_pair)
    #clf.predict([[0., 9.]])
    feature_pair_svm_predicted_values.append(clf.predict([feature_pair_for_measure_
accuracy[i]]))
```

In [17]:
```
print(feature_pair_svm_predicted_values)
```

```
[array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), arra
y([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
e=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uin
t8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8),
array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array
([0], dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0],
dtype=uint8), array([0], dtype=uint8), array([0], dtype=uint8), array([0], dtyp
```

Next, let's measure the accuracy of our prediction.

In [131]:

```
def Measure_accuracy(true_values, predictions):
    _sum_ = 0
    for truth, prediction in zip(true_values, predictions):
        if truth==prediction:
            _sum_+=1
    return _sum_/len(predictions)

Measure_accuracy(y_new, feature_pair_svm_predicted_values)
```

Out[131]: 0.7591904425539756

75% is the accuracy of our SVM prediction, according to the output above.

4.2 Compare your results to the random forest performance on the same task.

According to Machine Learning Master[x], Random Forest work well with a mixture of numerical and categorical features and use them as they are. While SVM relies on the distance between different points.

In this exercise, our Random Forest prediction was more accurate than SVM given that our data set is a mixture of numerical and categorical data. For a classification problem, such as this exercise, Random Forest gives the probability of belonging to class, while SVM gives the distance to the boundary.

REFERENCES

[1] Guide to Encoding Categorical Values in Python, Practical Business Python. URL: https://pbpython.com/categorical-encoding.html (https://pbpython.com/categorical-encoding.html)

[2] Bagging and Random Forest, Machine Learning Mastery. URL: https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/ (https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/)