Abstract

In this project, we were asked to experiment with a real world dataset, and to explore how machine learning algorithms can be used to find the patterns in data. We were expected to gain experience using a common data-mining and machine learning library, and were expected to submit a report about the dataset and the algorithms used. After performing the required tasks on a dataset of our choice, here in lies my final report.

Keywords: Machine Learning, Supervised learning,

Introduction:

Machine learning is a sub-domain of computer science which evolved from the study of pattern recognition in data, and also from the computational learning theory in artificial intelligence. It is the first-class ticket to most interesting careers in data analytics today. As data sources proliferate along with the computing power to process them, going straight to the data is one of the most straightforward ways to quickly gain insights and make predictions. Machine Learning can be thought of as the study of a list of sub-problems, viz: decision making, clustering, classification, forecasting, deep-learning, inductive logic programming, support vector machines, reinforcement learning, similarity and metric learning, genetic algorithms, sparse dictionary learning, etc. Supervised learning, or classification is the machine learning task of inferring a function from a labeled data . In Supervised learning, we have a training set, and a test set. The training and test set consists of a set of examples consisting of input and output vectors, and the goal of the supervised learning algorithm is to infer a function that maps the input vector to the output vector with minimal error. In an optimal scenario, a model trained on a set of examples will classify an unseen example in a correct fashion, which requires the model to generalize from the training set in a reasonable way. In

layman’s terms, supervised learning can be termed as the process of concept learning, where a brain is exposed to a set of inputs and result vectors and the brain learns the concept that relates said inputs to outputs. A wide array of supervised machine learning algorithms are available to the machine learning enthusiast, for example Neural Networks, Decision Trees, Support Vector Machines, Random Forest, Naïve Bayes Classifier, Bayes Net, Majority Classifier etc., and
They each have their own merits and demerits. There is no single algorithm that works for all cases, as merited by the No free lunch theorem . In this project, we try and find patterns in a dataset , which is a sample of e-bike price prediction and attempt to throw various intelligently-picked algorithms at the data, and see what sticks.

Problems and Issues :

Before we get started, we must know about how to pick a good machine learning algorithm for the given dataset. To intelligently pick an algorithm to use for a supervised learning task, we must consider the following factors :

1.Redundancy of Data:

If the data contains redundant information, i.e. contain highly correlated values, then it's useless to use distance based methods because of numerical instability. In this case, some sort of Regularization can be employed to the data to prevent this situation.

2.Dependent Features:

If there is some dependence between the feature vectors, then algorithms that monitor complex interactions like Decision Trees fare better than other algorithms.

3. Overfitting:

The programmer should know that there is a possibility that the output values may constitute of an inherent noise which is the result of human or sensor errors. In this case, the algorithm must not attempt to infer the function that exactly matches all the data. Being too careful in fitting the data can cause overfitting, after which the model will answer perfectly for all training examples but will have a very high error for unseen samples. A practical way of preventing this is stopping the learning process prematurely, as well as applying filters to the data in the pre-

learning phase to remove noises.

Only after considering all these factors can we pick a supervised learning algorithm that works for the dataset we are working on. For example, if we were working with a dataset consisting of heterogeneous data, then decision trees would fare better than other algorithms.

Dataset:

The dataset used is a sample of products of all e-bike (Price prediction of e-bikes).

The dataset that was used for this project is a subset

of a much larger dataset, and has the following feature vectors:

1. category
2. category inch
3. frame size
4. weight (lbs)
5. engine
6. engine position
7. engine power
8. mph
9. battery type
10. battery chara Wh
11. battery chara Ah
12. battery chara V
13. battery position
14. Gearshift
15. Gears
16. Gearshift Brand
17. Brakes Type
18. Brakes Model
19. Suspension front rear
20. Suspension brand
21. Suspension model
22. Permissible max weight lbs
23. Lighting
24. Rack
25. Fenders
26. Company Name
27. Price
28. Model year
29. Frame type
30. Ebike type

Data types of variables:

```
df.dtypes
```

```
category                    object
category_inch               object
frame_size                  object
weight (lbs)               float64
engine                      object
engine_position             object
engine_power               float64
mph                          int64
battery_type                object
battery_chara_Wh           float64
battery_chara_Ah           float64
battery_chara_V            float64
battery_position            object
Gearshift                   object
Gears                      float64
Gearshift_Brand             object
Brakes_Type                 object
Brakes_Model                object
Suspension_front_rear       object
Suspension_brand            object
Suspension_model            object
Permissible_max_weight_lbs  object
Lighting                    object
Rack                        object
Fenders                     object
Company_Name                object
Price                       object
Model_year                   int64
Frame_type                  object
ebike_type                  object
```

Missing value Imputation:

Most of the feature vectors had large amount of missing values like:

```
df.isnull().mean()
```

```
category                    0.406614
category_inch               0.000000
frame_size                  0.470890
weight (lbs)                0.608291
engine                      0.004658
engine_position             0.010247
engine_power                0.022823
mph                         0.000000
battery_type                0.004658
battery_chara_Wh            0.004658
battery_chara_Ah            0.004658
battery_chara_V             0.004658
battery_position            0.023288
Gearshift                   0.001863
Gears                       0.079646
Gearshift_Brand             0.055892
Brakes_Type                 0.000000
Brakes_Model                0.067070
Suspension_front_rear       0.000000
Suspension_brand            0.387517
Suspension_model            0.391244
Permissible_max_weight_lbs  0.000000
Lighting                    0.000000
Rack                        0.000000
Fenders                     0.000000
Company_Name                0.000000
Price                       0.000000
Model_year                  0.000000
Frame_type                  0.000000
ebike_type                  0.000000
```

Using Simple Imputer we try to impute those missing values like:

```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
#df['model_year'] = imp.fit_transform(df[['model_year']])
df1['category'] = imp.fit_transform(df1[['category']])


#df['ebike_type'] = imp.fit_transform(df[['ebike_type']])
#df['frame_type'] = imp.fit_transform(df[['frame_type']])
df2['engine'] = imp.fit_transform(df2[['engine']])
df2['engine_position'] = imp.fit_transform(df2[['engine_position']])

df2['battery_type'] = imp.fit_transform(df2[['battery_type']])
df2['battery_position'] = imp.fit_transform(df2[['battery_position']])

df2['Gearshift_Brand'] = imp.fit_transform(df2[['Gearshift_Brand']])
df2['Gearshift'] = imp.fit_transform(df2[['Gearshift']])

df2['Brakes_Model'] = imp.fit_transform(df2[['Brakes_Model']])
df2['Suspension_brand'] = imp.fit_transform(df2[['Suspension_brand']])
df2['Suspension_model'] = imp.fit_transform(df2[['Suspension_model']])
df2['Permissible_max_weight_lbs'] = imp.fit_transform(df2[['Permissible_max_weight_lbs']])
```

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')

df2['category_inch'] = imp.fit_transform(df2[['category_inch']])
df2['mph'] = imp.fit_transform(df2[['mph']])
df2['weight'] = imp.fit_transform(df2[['weight']])

df2['engine_power'] = imp.fit_transform(df2[['engine_power']])

df2['battery_chara_Wh'] = imp.fit_transform(df2[['battery_chara_Wh']])
df2['battery_chara_Ah'] = imp.fit_transform(df2[['battery_chara_Ah']])
df2['battery_chara_V'] = imp.fit_transform(df2[['battery_chara_V']])

df2['Gears'] = imp.fit_transform(df2[['Gears']])
```

Correlation matrix

	category_inch	weight	engine_power	mph	battery_chara_Wh	battery_chara_Ah	battery_chara_V	Gears	Model_year
category_inch	1.000000	-0.384173	-0.006035	0.030394	0.124143	0.070237	-0.012202	0.198959	0.156182
weight	-0.384173	1.000000	0.078564	0.087083	0.090092	0.068840	0.032705	-0.154418	0.065835
engine_power	-0.006035	0.078564	1.000000	0.362689	0.092101	-0.001848	0.039668	0.038586	-0.082710
mph	0.030394	0.087083	0.362689	1.000000	0.148855	0.099763	0.047016	0.083633	0.066728
battery_chara_Wh	0.124143	0.090092	0.092101	0.148855	1.000000	0.627624	0.302423	0.112640	0.350813
battery_chara_Ah	0.070237	0.068840	-0.001848	0.099763	0.627624	1.000000	0.790842	0.082073	0.289652
battery_chara_V	-0.012202	0.032705	0.039668	0.047016	0.302423	0.790842	1.000000	-0.011364	0.140987
Gears	0.198959	-0.154418	0.038586	0.083633	0.112640	0.082073	-0.011364	1.000000	0.044844
Model_year	0.156182	0.065835	-0.082710	0.066728	0.350813	0.289652	0.140987	0.044844	1.000000

Result:

1.Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
result = model.fit(X_train, Y_train)
```

```
y_pred= model.predict(X_test)
```

```
train_MSE = np.mean((Y_train - result.predict(X_train))**2)
test_MSE = np.mean((Y_test - result.predict(X_test))**2)
print(np.sqrt(train_MSE))
print(np.sqrt(test_MSE))
print('Variance score: %.2f' % model.score(X_test, Y_test))
print('Variance score: %.2f' % model.score(X_train, Y_train))
```

```
1262.1223106456875
1365.8860071055653
Variance score: 0.31
Variance score: 0.35
```

2. Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

regressor = DecisionTreeRegressor(random_state = 10)
result=regressor.fit(X_train, Y_train)
y_pred = regressor.predict(X_test)
train_MSE = np.mean((Y_train - result.predict(X_train))**2)
test_MSE = np.mean((Y_test - result.predict(X_test))**2)
print(np.sqrt(train_MSE))
print(np.sqrt(test_MSE))
print('Variance score: %.2f' % regressor.score(X_test, Y_test))
print('Variance score: %.2f' % regressor.score(X_train, Y_train))
```

```
109.00978814814012
1169.157784158171
Variance score: 0.49
Variance score: 1.00
```

3. Ada boost

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import RandomizedSearchCV
```

```
param_dist = {'n_estimators': [50, 100],
              'learning_rate' : [0.01,0.05,0.1,0.3,1],
              'loss' : ['linear', 'square', 'exponential', 'huber']
             }
```

```
pre_gs_inst = RandomizedSearchCV(AdaBoostRegressor(),
                                 param_distributions = param_dist,
                                 cv=3,
                                 n_iter = 10,
                                 n_jobs=-1)
pre_gs_inst.fit(X_train, Y_train)
```

```
RandomizedSearchCV(cv=3, estimator=AdaBoostRegressor(), n_jobs=-1,
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1, 0.3,
                                                          1],
                                        'loss': ['linear', 'square',
                                                 'exponential', 'huber'],
                                        'n_estimators': [50, 100]})
```

```
pre_gs_inst.best_params_
```

```
{'n_estimators': 100, 'loss': 'exponential', 'learning_rate': 0.1}
```

```
import copy
ada_best = copy.deepcopy(pre_gs_inst.best_params_)
```

```
rs_ada = AdaBoostRegressor(**ada_best)
rs_ada.fit(X_train, Y_train)
```

```
AdaBoostRegressor(learning_rate=0.1, loss='exponential', n_estimators=100)
```

```
y_pred = rs_ada.predict(X_test)
from sklearn.metrics import r2_score, mean_absolute_error
```

```
print(r2_score(Y_test, y_pred))
print(np.sqrt(mean_absolute_error(Y_test, y_pred)))
print((np.abs(Y_test - y_pred)/Y_test).mean())
rmse = np.sqrt(mean_squared_error(Y_test, y_pred))
print("RMSE: %f" % (rmse))
```

```
0.454299646219254
29.316285871321128
0.24358633134952515
RMSE: 1213.941902
```

4. Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rft = RandomForestRegressor()
rft.fit(X_train, Y_train)
y_pred = rft.predict(X_test)
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
print(np.sqrt(mean_squared_error(Y_test, y_pred)))
print(r2_score(Y_test, y_pred))
print( mean_absolute_error(Y_test, y_pred))
```

```
893.8790699432658
0.7041197564563688
492.16013571399276
```

```
print('Variance score: %.2f' % rft.score(X_test, Y_test))
print('Variance score: %.2f' % rft.score(X_train, Y_train))
```

```
Variance score: 0.70
Variance score: 0.95
```

5. Random Forest Regressor with parameter tuning

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [50, 60, 80, 90, 100],
    'max_features': [0.3, 0.5, 'sqrt'],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [ 2, 4, 5, 8],
    'n_estimators': [200]
}

regr = RandomForestRegressor(random_state=0,n_jobs=-1)

clf = GridSearchCV(regr, param_grid)
clf.fit(X_train, Y_train)
```

```
GridSearchCV(estimator=RandomForestRegressor(n_jobs=-1, random_state=0),
             param_grid={'bootstrap': [True],
                         'max_depth': [50, 60, 80, 90, 100],
                         'max_features': [0.3, 0.5, 'sqrt'],
                         'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [2, 4, 5, 8],
                         'n_estimators': [200]})
```

```
y_pred = clf.predict(X_test)
```

```
print(np.sqrt(mean_squared_error(Y_test, y_pred)))
```

```
926.4763662426933
```

```
print('Variance score: %.2f' % clf.score(X_test, Y_test))
print('Variance score: %.2f' % clf.score(X_train, Y_train))
```

```
Variance score: 0.68
Variance score: 0.83
```