Evaluation of Different Classification Algorithms using scikitLearn

Khaled Abdelaal
kha217@lehigh.edu
Lehigh University

ISE364 Final Project

12/12/2019

Contents

List of Figures

List of Tables

Abstract

In this technical report, different machine learning models are used to classify some target data by training and fitting these models using some labelled training data. This data is firstly pre-processed. Missing values in each column are replaced by the mode of the values in that column. Then, categorical data in feature columns are encoded using one-hot encoding in order for the model to be able to accurately operate on them. All input data is then scaled using a standard scaler. Next, data is split into two parts: training data and test data. The model of choice is fit using training data, predictions are then calculated and compared to the labels in the test data. Accuracy of several models is reported and sensitivity to parameters is also studied on some models. The models evaluated in this reports are: Linear Regression, Logistic Regression, KNN (K-Nearest Neighbors), Decision Trees, Random Forests, SVM (Support Vector Machines), and MLP (Multi-Layer Perceptron).

1 Introduction

Machine Learning is an aspect of AI (Artificial Intelligence) that provides systems with the capability of learning from experience without being explicitly programmed. It allows systems to learn from data and patterns on their own. Many applications benefit from Machine Learning techniques in different fields such as: business intelligence, stocks market predictions, image detection and recognition, natural language processing, and many more. The main idea of machine learning algorithms is to iteratively learn from data to conclude insights about the inherent relationships between different features and targets. A model is developed based on these inferred relationships, then this model is tested for accuracy. Depending on the accuracy, the learning process might have more iterations until an acceptable accuracy is reached. After that, the model can be used to predict new data which the model has not been trained on.

There are a few types of machine learning algorithms: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the model already has labelled data on which it trains and tests. Then, the model's main job is to predict new labels for new, unseen data based on the features which are known to it. Classification is one of the fundamental supervised machine learning problems and is the focus of this report. Another type of algorithms is the unsupervised learning. In this category, the goal is to group or "cluster" a set of unlabelled data points together, based on features. One more popular type of algorithms is reinforcement learning. In this type, the objective is to solely use experience in the learning process. The system improves accuracy by taking more actions and gaining more experience.

2 Problem Definition

As mentioned, supervised learning (and more specifically classification algorithms) is the focus of this report. The problem in hand is: given a data set `data.csv` with 15 different columns and no headers. Columns 0, 2, 4, 10, 11, and 12 are numeric data while columns 1, 3, 5, 6, 7, 8, 9, and 13 are categorical data. Column 14 has labels (target column) which are categorical values as well, either SMALL or LARGE. A classification model is required to be developed using data in `data.csv`. This model is then supposed to predict target column in some other data set `futures.csv`. Both files have some missing values marked by the question mark '?'. The main job is to create a classification model using data in `data.csv` and then use the developed model on the data contained in `futures.csv` in order to predict the target values for each row (either SMALL or LARGE).

3 Proposed Approach

In order to accomplish the task in hand, a few steps are required. In this section, all of these steps are described in enough details.

3.1 Exploring Data

In this project, a python library "pandas" [3] is used for data analysis and manipulation. Before working with data, it might be useful to explore it by looking at the first few rows using `dataframe.head()` method. This command outputs the first few rows, so that we can see the nature of the values we are dealing with. Also, it could be handy to find out some summary regarding numeric data, such as count, mean, standard deviation, etc. We can do that using the `dataframe.describe()` method. Moreover, using seaborn [1] which is a statistical data visualization framework, we can quickly discover relationships between different numeric fields in the dataframe. Figure 1 shows such figure. It is a pairplot of the dataframe, with a specific hue on the target data column (in our case column 14), to show different levels of that categorical variable by the color of the plot element. We can also check a correlation heat map to find out if there are any obvious correlations between different columns. Seaborn and pandas also provide this kind of plots using `seaborn.heatmap(dataframe.corr())`. Figure 2 shows this correlation using heatmap on the training dataset used in this project.

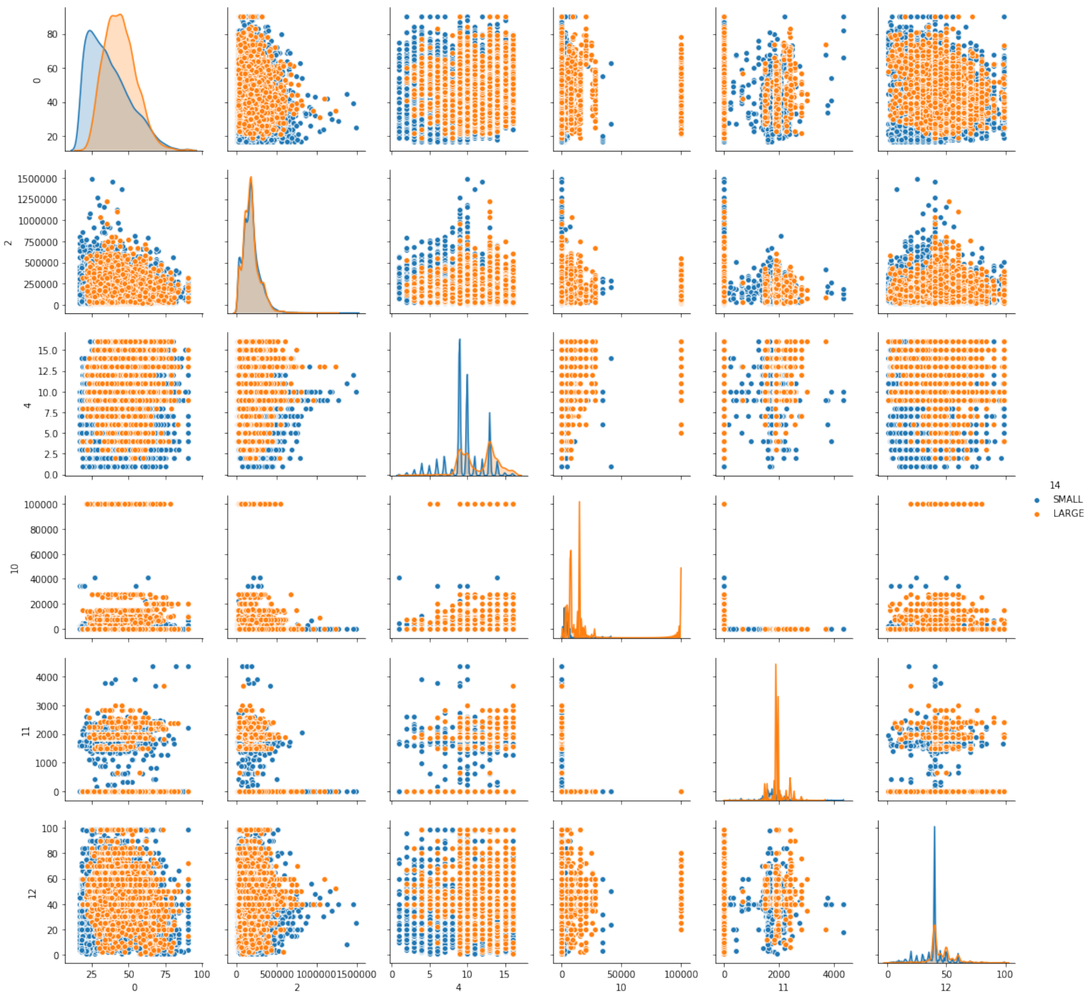

Figure 1: A seaborn pair-plot of the training data. Plot is hued using the target column (column 14) which has values of either SMALL or LARGE

3.2 Choosing a Machine Learning Framework

First, a machine learning framework is chosen in order to accelerate the implementation process. The framework of choice was ScikitLearn [4]. The main reasons for choosing ScikitLearn are its simplicity and

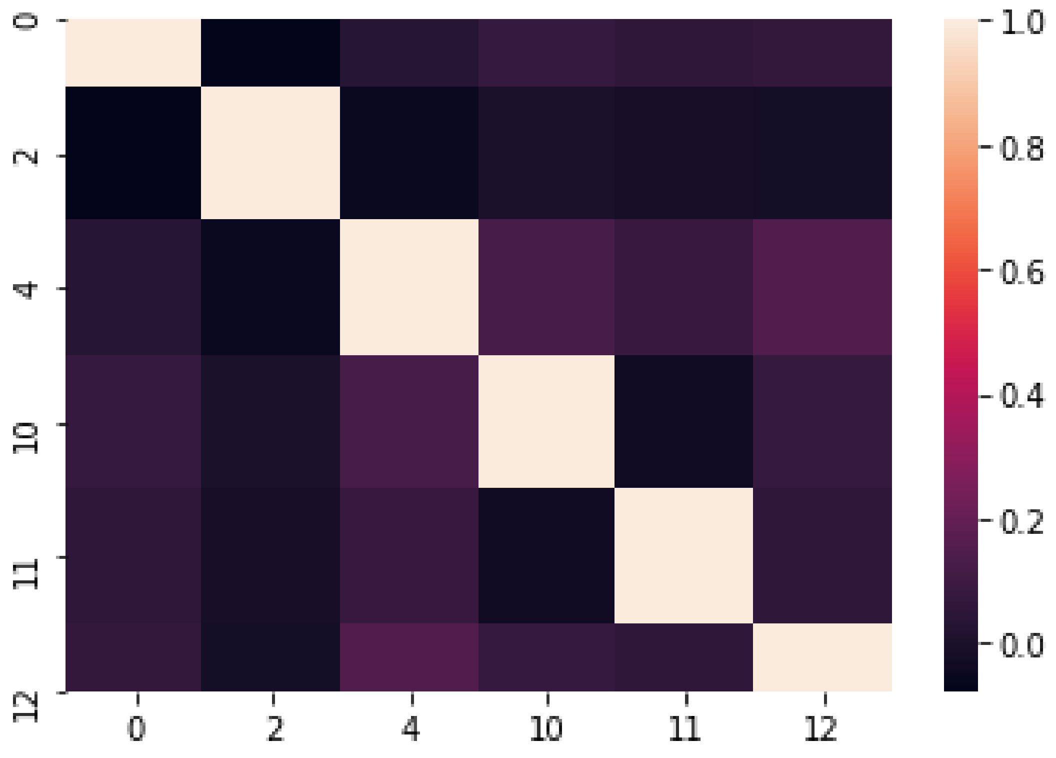

Figure 2: A seaborn heatmap showing the correlation between different columns in the training data set. Numbers on the horizontal andd vertical axes correspond to column numbers (0 to 14) in the training dataset. Note that only numeric columns are included

strength. ScikitLearn is powerful enough to implement different models for the problem being addressed, yet it is simple and does not require lots of complexity.

3.3 Data Pre-processing

This step might be the most important and effort consuming step in solving such a problem. In this step, data should be cleaned and prepared in order to make it suitable to be fed into a machine learning model. In this report, three different steps are seen important as part of the data pre-processing step.

3.3.1 Dealing with missing values

As mentioned before, `data.csv` and `futures.csv` have some missing values. Machine learning models (or statistical models in general) cannot deal with incomplete sets. In order to handle the cases where missing values in the data set exist, several methods have been proposed in literature [2]. One of them is to discard the whole row which contains a missing value. Another commonly used method is imputation. In imputation, missing values are replaced with some other value that represents the column. One popular, commonly accepted metric, of column value representation is the mode of the values in that column. In this report, imputation using mode for handling missing values is used.

3.3.2 Converting Categorical Data into Numeric Values

Most practical data sets would include categorical values. Categorical values are those which are not numeric and represent a specific category. For example, a dataset might have a column capturing the model of a specific vehicle. Vehicle models are not naturally numeric, we need textual representation to indicate which model this vehicle is. However, most machine learning algorithms cannot work directly with categorical data. For this reason, this kind of data should be converted somehow into a numeric representation in order for a machine learning algorithm to be able to operate on.

There are different methods to do that conversion. One method is to define a set of corresponding numeric values for the categorical values we have. Then, each occurrence of a categorical value is replaced by its numeric corresponding value. This can be simply done in pandas using the `replace()` method and defining dictionaries that map categorical values into numeric ones.

One other common method is one-hot encoding. In this method, each category is converted into a separate column, each column could have a value of 0 or 1. For each row, only the column corresponding to the specific categorical value will have a value of 1, while the rest of the category columns will have a value of 0. One-hot encoding could be done in pandas using the `pandas.get_dummies()` method. This will allow each feature to be weighted the same (not improper weights to values, which might be the case in the previous replace method), but on the other hand it much add much more columns. In this project, we are using one-hot encoding for features, and binary value replacement for the target column (0 for SMALL and 1 for LARGE).

3.3.3 Feature Scaling

Features have widely sparse values which have completely different ranges. This affects model's decisions since features with high values might dominate and highly control final decision. In order to make sure that all values are within a standard length, scaling should be performed on data before feeding it into the model. For scaling, a `sklearn.preprocessing.StandardScaler` was used. This scaler scores each sample value x using the following equation:

$$z = (x - u)/s$$

where z is the standard score for x, u is the mean of the training samples, and s is the standard deviation of the samples.

3.4 Preparing Data for the Machine Learning Model

Whenever a model is being developed, we start with some data to train the model on. This set of data is called the training data. In this project, the training data is `data.csv`. However, a method is needed in order to evaluate the model and report the accuracy of that model. Since the only data set with labels that we have access to is the training data, we can split the training data into two parts in order to evaluate the system. One part is the training data and the other part is testing data. The training data part is used for the original purpose of training the machine learning model. After training the model, the values for the target column is predicted for the test portion of the data set. Then, these predicted values are compared to the original labels (since we already have them) to assess the accuracy of the model. scikitlearn provides the necessary tools in order to perform this task. The `train_test_split()` method in `sklearn.model_selection` provides the ability to split a data set into two parts: training data and test data. One could specify the percentage of test data out of the whole data set. We use this method to train and evaluate our models.

3.5 Choosing a Machine Learning Model

As mentioned in section 2, the goal of this project to predict the values for a target column in `futures.csv` based on the model trained on `data.csv`. For that, we need to choose a specific machine learning model and use the training data set to train and test the model. For any classification problem, there are many candidate models that could be used. These models include, but not limited to, Logistic Regression, KNN, SVM, Decision Trees, Random Forests, and MLP. We evaluate all of these models on the training data set, and then use the one with the highest accuracy on the target data set.

3.5.1 Logistic Regression

Logistic regression is another machine learning model for binary classification of data. It is suitable for the question of classifying data into two different categories. It could also be extended to classify data against multiple classes (not only two). It basically depends on the logit function which is the logarithm of the probability of a positive event happening (the log of the odd ratio). Logistic regression model is available for direct use in scikit learn and is bundled as part of the `sklearn.linear_model.LogisticRegression`.

3.5.2 K-Nearest Neighbors (KNN)

KNN is another classification algorithm that classifies data points based on distance. Whenever a new data point is to be classified, KNN calculates distance between that point and all other existing points. It then sorts distances in an ascending order. Finally, it predicts the new class to be the same class of the majority of the K closest points. As could be imagined, setting the value for K is a crucial decision to be taken and could significantly affect the accuracy of the prediction of the model. Some of the advantages of this algorithm is that it is simple to implement and can easily work with any number of classes. It also has few parameters to tune (value for K, and which distance metric to use for example Euclidean vs Manhattan distance). On the other hand, it might introduce a significant overhead for large datasets with high misprediction penalty. KNN is also available in scikitlearn `sklearn.neighbors.KNeighborsClassifier` class.

3.5.3 Support Vector Machine (SVM)

SVM is a classification algorithm that tries to solve this problem by directly finding a plane that separates the two different classes in the feature space. In other words, based on number of features, a hyperplane is found that could approximately separate features values. In a two-dimensional space, a hyperplane is a line dividing the plane into two parts, where each class lies in one of those parts. Some of the important parameters to tune in SVM are the regularization parameter (C), gamma, and kernel. C represents the required accuracy of classification, or how much the optimizer should avoid misclassifying data. For large values of C, the classifier will choose a hyperplane with small margin. For small values of C, a hyperplane with a large margin will be chosen. For gamma, it defines how far the influence of a single training example reaches. If the value of gamma is low, it means that the influence will extend to far away points. However, when the value of gamma is high, it means that the influence will be limited to nearby points. The kernel is the function transforming features from some dimensional space to another and back to the original one. SVM model is also available in the scikitlearn library through the `sklearn.svm.SVC` class.

3.5.4 Decision Trees

A decision tree is some tree in which each internal node represents a test on a specific feature and each leaf represents final label (or target class). The construction of decision trees is done by finding a way to split features based on some conditions.

3.5.5 Random Forests

Random forests are an extension to decision trees. In random forests, multiple trees are used for training and prediction. The majority of trees vote for the final label decision. One important parameter while dealing with random forests is the number of estimators, or in other words, the number of trees within the forest.

3.5.6 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron is a Neural Network (NN) algorithm which includes various layers. There is one input layer and one output layer, each containing a fixed number of neurons depending on the number of

	precision	recall	f1-score	support
0	0.88	0.93	0.91	7436
1	0.73	0.60	0.66	2333
accuracy			0.85	9769
macro avg	0.81	0.77	0.78	9769
weighted avg	0.85	0.85	0.85	9769

Table 1: Classification Report for Logistic Regression Model

features and target values in the data set. A neuron does the processing (calculating weighted sum) of the neural network. An activation function is used to determine whether a neuron should be fired (triggered) or not. There are many different activation functions such as: step, sigmoid, tanh and ReLU. A number of hidden layers (i.e, layers that lie between input and output layers) can be defined as well. Within each hidden layer, the number of neurons could be defined. There is no single correct setting for the number of layers and/or the number of neurons per layer. One other important parameter is the maximum number of iterations (or epochs) during which the model will keep training.

4 Evaluation and Results

In this section, evaluation methods and steps are discussed. All results obtained during the implementation and the evaluation of the models (and parameter tuning within each model) are reported.

4.1 Evaluation Methodology

The evaluation of the accuracy of each model is done using confusion matrix and classification report. Predicted data is tested against the labels in the test part of the data. These evaluation metrics are available in scikitlearn as `sklearn.metrics.confusion_matrix` and `sklearn.metrics.classification_report`

4.2 Data Pre-processing

For all columns, missing data were replaced by the `mode()` value of each column. Categorical features were converted into numeric ones using one-hot encoding (`pandas.get_dummies()`. The label column was replaced by a value of 0 for SMALL and a value of 1 for LARGE. Features were then scaled using `sklearn.preprocessing.StandardScaler`. Eventually, training data was split using the `train_test_split()` method. The size of the test part was set to be 30% of the whole training data set. A random seed was used to ensure that random samples are drawn out of the training set to be selected in the test set.

4.2.1 Logistic Regression Model

A logistic regression model was instantiated with the default parameters. A `'liblinear'` solver was used. Then, the classification report was printed and predicted values compared against test data labels. Classification report for the logistic regression model is shown in table 3. It could be noticed that the accuracy of that model using default parameters is around 85%.

4.2.2 KNN

For KNN, we identify one critical parameter, which is K or the number of neighbors to consider. In our evaluation, values of K from 1 to 100 were used, model was fit for each value, targets were predicted and mean error was calculated for each trial. This evaluation method is done in order to choose the optimal value for K which would yield the least mean error and then use it for the final model. Figure 3 shows the

	precision	recall	f1-score	support
0	0.87	0.92	0.89	7436
1	0.68	0.56	0.62	2333
accuracy			0.83	9769
macro avg	0.78	0.74	0.76	9769
weighted avg	0.83	0.83	0.83	9769

Table 2: Classification Report for KNN Model with K=33

relationship between the value of K on the horizontal axis and the mean error rate on the vertical axis. We can see that a value of K=33 yields the lowest mean error rate. So, this value is used for the KNN model and the classification report we get for that value is shown in table 2. From the classification report, we notice an average accuracy of 83% for the KNN model with K=33.

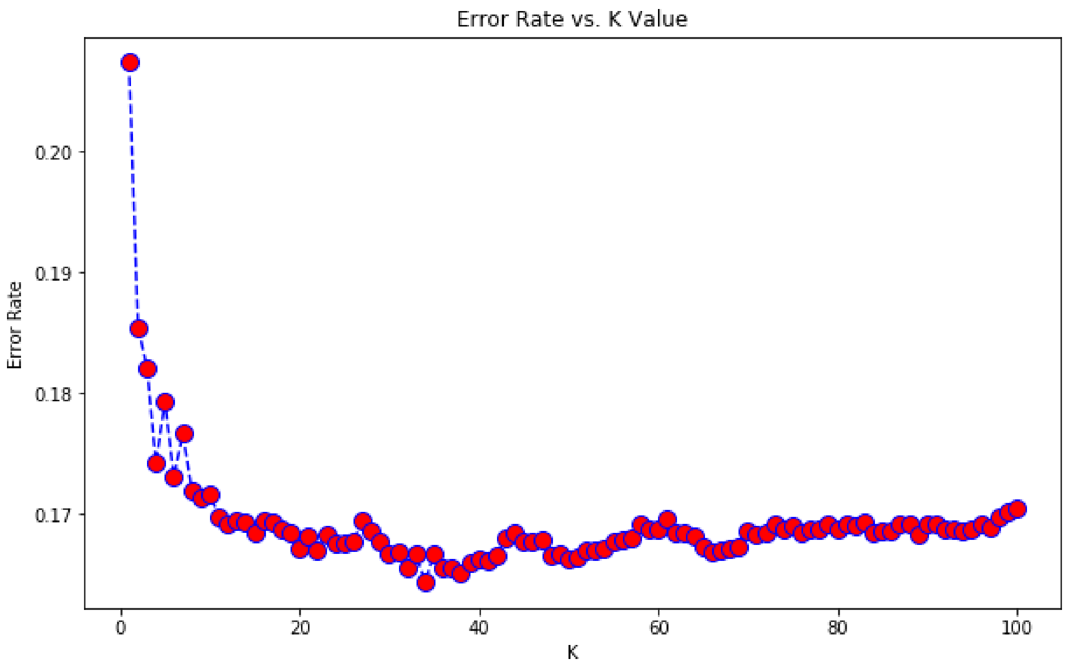

Figure 3: Evaluation of different number of estimators (on the horizontal axis) and finding mean error rate (vertical axis) for a random forest model

4.2.3 Decision Tree

A decision tree model was instantiated using the default parameters. The criterion used was entropy. Table **??** shows the classification report for that model. One can notice that average accuracy for that model on the training data is around 82%.

4.2.4 Random Forests

Going further, a random forests model was evaluated as well. Different number of estimators were evaluated, varying from 100 to 1000. Same criterion (entropy) was used for all. Figure 4 shows the effect of changing the number of estimators on the mean error rate. From that figure, it is noticed that a value of 500 for the number of estimators yields least mean error rate on this data set. So, this number of estimators

	precision	recall	f1-score	support
0	0.88	0.88	0.88	7436
1	0.61	0.62	0.62	2333
accuracy			0.82	9769
macro avg	0.75	0.75	0.75	9769
weighted avg	0.82	0.82	0.82	9769

Table 3: Classification Report for Decision Tree Model

is used and the classification report is obtained as shown in table 4. From the classification report, it is noticed that average accuracy for this random forest model is around 86%.

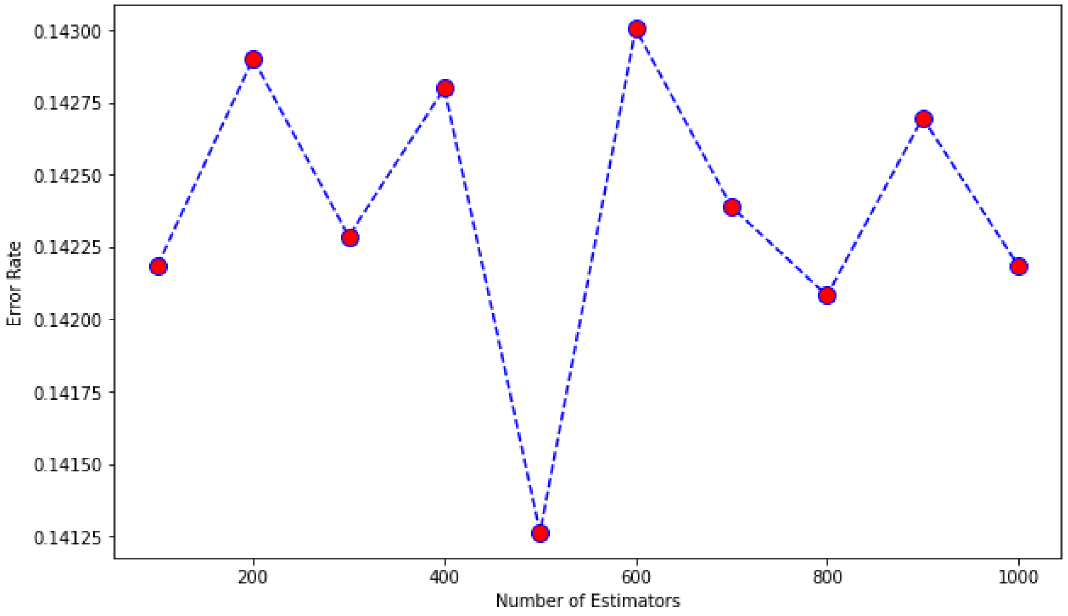

Figure 4: Evaluation of different number of estimators (on the horizontal axis) and finding mean error rate (vertical axis) for a random forest model

4.2.5 Support Vector Machines (SVM)

SVMs are employed as an additional model to train and test the data set. There are several parameters that could be tuned. For this reason, we use grid search in order to find the optimum set of parameters among the given set. We try values of C of 0.1, 1, 10, 100, values of gamma of 1, 0.1, 0.01, 0.001 all with kernel `'rbf'`. The grid search exhaustively search all of the different possible combinations. Results show that the configuration of C=10, gamma=0.001 with kernel='rbf' yields the best results among the tested combinations. Predictions for that combination is obtained and classification report is shown in table 5. It can be seen that this model achieves an accuracy of around 86%.

4.2.6 Multi-Layer Perceptron

MLP model was also evaluated with a few different configurations that varies in the number of hidden layers, number of neurons in each layer and maximum number of iterations (number of epochs). Finally, we use two hidden layers with 30 neurons each and maximum iterations of 1000 iterations. An accuracy of around 83% was obtained according to the classification report shown in table 6.

	precision	recall	f1-score	support
0	0.89	0.93	0.91	7436
1	0.74	0.64	0.68	2333
accuracy			0.86	9769
macro avg	0.81	0.78	0.80	9769
weighted avg	0.85	0.86	0.85	9769

Table 4: Classification Report for Random Forest Model with 500 ESTIMATORS

	precision	recall	f1-score	support
0	0.88	0.94	0.91	7436
1	0.75	0.60	0.66	2333
accuracy			0.86	9769
macro avg	0.82	0.77	0.79	9769
weighted avg	0.85	0.86	0.85	9769

Table 5: Classification Report for SVM with C=10, gamma=0.001 and kernel='rbf'

	precision	recall	f1-score	support
0	0.89	0.88	0.89	7436
1	0.63	0.65	0.64	2333
accuracy			0.83	9769
macro avg	0.76	0.77	0.76	9769
weighted avg	0.83	0.83	0.83	9769

Table 6: Classification Report for MLP with 2 hidden layers of 30 neurons each and 1000 iterations

4.3 Predicting Target Values for `futures.csv`

In order to eventually predict the target values for the given data set, a number of steps should be followed. Some of those steps depend on previously discussed methods. Here, different steps required to achieve that goal are enumrated

4.3.1 Choosing the Right Model

According to the previous discussion, both SVM with C=10, gamma=0.001 and kernel='rbf', and Random Forest with 200 estimators achieve highest accuracy among evaluated models and configurations (86%). A decision to use one of them on `futures.csv` should be made. Random Forests were chosen due to their simplicity and few important parameters.

4.3.2 Preparing Data

In order for the data to be readable by the model, same procedure used in section 3.4 is applied. Replacing missing data, converting categorical values into numeric one and scaling data was repeated on the new data set.

4.3.3 Feeding data to model

In this case, we do not have to split data set into train and test, because we do not have labels. The task is to find out the labels. So, after preparing data, the whole data is fed into the random forest model which has been previously trained. Then we get predicted values.

4.3.4 Saving Predicted Data

Predicted values would be a pandas array, in which each element has a value of 1 or 0. First, we concatenate this array to the original dataframe (before data preparation). After that, we do the reverse replacement. This time, we replace each instance of 0 with the categorical value "SMALL" and each occurence of 1 with the categorical value "LARGE". After that, we can use pandas `DataFrame.to_csv()` method in order to save the newly generated data frame with predicted labels, to a csv file. We should also set the parameters `header=None, index=None`. Now, we have a csv file containing the original content of `futures.csv` concatenated to a column of predicted labels.

5 Summary and Conclusions

In this report, we evaluated different Machine Learning models on a specific training data set (`data.csv`). We first explored the clean-up and preparation of data since it is a vital step in developing machine learning solutions. Then, we moved on into evaluating the actual model and tuning its parameters to get the least mean error rate (or highest accuracy). The algorithms we evaluated, which are different classification algorithms, are Logistic Regression, K-Nearest Neighbors, Decision Tress, Random Forest, Support Vector Machines, and Multi-layer Perceptron. For this specific data set and with the specific parameters evaluated, highest accuracy was obtained using a variant of Random Forest. We used the same configuration and applied it to the test data set (`futures.csv`). In order to predict the labels for that data set, we went through data preparation and then feeding the correct format of data to the Random Forest model. We got the labels in the form of numeric values, which had to be converted into the original categorical values (SMALL or LARGE). Then, this column was concatenated to the original data and saved as a new csv (`predicted_futures.csv`)

References

[1] Seaborn: statistical data visualization framework. https://seaborn.pydata.org/.

[2] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, Mar 2010.

[3] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A Appendix A: Source code for the project

Listing 1: Source code for project

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df = pd.read_csv('data.csv', header=None)
df.head()
df.describe()
sns.pairplot(df, hue=14)

#Replacing missing values with the mode() of the corresponding column
for i in range(14):
    df[i].replace(to_replace ='?',value=df[i].mode()[0], inplace=True)

#Finding out which columns are numeric and which are categorical
df.info()

#Convert categorical features into numeric values using one-hot encoding
   then converting target values into binary (0 or 1) using replace
#Categorical Columns (except for target)
col_idx = [1,3,5,6,7,8,9,13]
converted_cols = []
for i in col_idx:
    converted_cols.append(pd.get_dummies(df[i], drop_first=True))
    df.drop(i, axis=1, inplace=True)
converted_cols.append(df)
df.replace({14:{"SMALL":0,"LARGE":1}}, inplace=True)

#Concatenate all data back again
df = pd.concat(converted_cols,axis=1)

#Scale feature values using standard scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop(14, axis=1))
scaled_features = scaler.transform(df.drop(14, axis=1))

#Split data into train and test parts
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df[14],
   test_size=0.3,random_state=101)

#Now, start with first algorithm, Logistic Regression
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression(solver='liblinear')
logmodel.fit(X_train,y_train)
pred_log = logmodel.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred_log)
from sklearn.metrics import classification_report
print(classification_report(y_test,pred_log))

#Evaluate the next algorithm, KNN
from sklearn.neighbors import KNeighborsClassifier


#Find a good value for K using brute-force by trying all values in the range
    1 to 100 as K, then find error rate for each K value
error_rate = []
for i in range(1,101):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
print("Done!")

#Plot Error Rate vs. Different K values
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(range(1,101),error_rate,color='blue', linestyle='dashed', marker='o
    ',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')

error_rate.index(min(error_rate))
#We found out that the best K value (least error mean error rate) is 33
#Use K=33 to predict and find accuracy
knn = KNeighborsClassifier(n_neighbors=33)

knn.fit(X_train,y_train)
pred_knn = knn.predict(X_test)

print('WITH K=33')
print('\n')
print(confusion_matrix(y_test,pred_knn))
print('\n')
print(classification_report(y_test,pred_knn))

#Move on to the next algorithm, Decision Trees
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='entropy')
dtree.fit(X_train,y_train)
```

```
pred_dtree = dtree.predict(X_test)
print(classification_report(y_test,pred_dtree))

#Now, try random forests
#Also try different n_estimators (from 100 to 1000) to find out which value
   yields least mean error rate
from sklearn.ensemble import RandomForestClassifier
error_rate_rf = []
for i in range(1, 11):
    rfc = RandomForestClassifier(n_estimators=i*100, criterion='entropy')
    rfc.fit(X_train,y_train)
    rfc_pred_i = rfc.predict(X_test)
    error_rate_rf.append(np.mean(rfc_pred_i != y_test))
print("Done!")

#Plot error vs. n_estimators
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(range(100,1100,100),error_rate_rf,color='blue', linestyle='dashed',
    marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error␣Rate␣vs.␣Number␣of␣Estimators')
plt.xlabel('Number␣of␣Estimators')
plt.ylabel('Error␣Rate')

#We can see that 500 is the best among these values
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=500, criterion='entropy')
rfc.fit(X_train, y_train)

rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test,rfc_pred))
print(classification_report(y_test,rfc_pred))

#Try the next algorithm, Support Vector Machines
#Use GridSearch to find a good configuration for C and gamma
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel':
   ['rbf']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
grid.fit(X_train,y_train)
grid.best_params_
grid.best_estimator_
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))

#Last algorithm to try, MLP
```

```
from sklearn.neural_network import MLPClassifier
#Two hidden layers of 30 neurons each, for 1000 epochs
mlp_model = MLPClassifier(hidden_layer_sizes=(30,30),max_iter=1000)
mlp_model.fit(X_train,y_train)
pred_mlp = mlp_model.predict(X_test)
print(confusion_matrix(y_test,pred_mlp))
print(classification_report(y_test,pred_mlp))

#We're done evaluating all models, we decided to use Random Forest on
    futures.csv
Now, we load futures.csv into df_new
df_new = pd.read_csv('futures.csv', header=None)
df_new.info()
#Keep a copy of the original dataframe, because we'll need it later
df_futures_org = df_new.copy()

#Replace missing values in futures.csv
for i in range(13):
    df_new[i].replace(to_replace ='?',value=df_new[i].mode()[0], inplace=
        True)

#One-hot encoding for categorical features
col_idx_n = [1,3,5,6,7,8,9,13]
converted_cols_n = []
for i in col_idx_n:
    converted_cols_n.append(pd.get_dummies(df_new[i], drop_first=True))
    df_new.drop(i, axis=1, inplace=True)
converted_cols_n.append(df_new)

df_new = pd.concat(converted_cols_n,axis=1)

#Scale features using standard scaler
scaler_n = StandardScaler()
scaler_n.fit(df_new)
scaled_features_n = scaler_n.transform(df_new)

#Now, we can predict the values for the target column using the random
    forest model (which was fit previously)
futures_pred = rfc.predict(scaled_features_n)

#Add the predicted column to the original dataframe version
df_futures_org[14] = futures_pred.tolist()

#Now, we need to convert numeric values for the labels (0 or 1) back into
    categorical values (SMALL or LARGE)
df_futures_org.replace({14:{0:"SMALL",1:"LARGE"}}, inplace=True)

#Last step, save the dataframe which contains original data + the newly
    predicted column to a csv file
df_futures_org.to_csv('predicted_futures.csv', header=None, index=None)
```