Project CS 556

The project is worth 100 points in total. All teams need to have 3 members.   
  
**What to submit:**

1. You will be provided with a jupyter notebook to give your requisite code / analysis.
2. In addition, you can use this word document to answer all the non-coding questions (inline) and submit a single document for all the analysis. (Questions will mostly be localized to the “Discussion” section of each question.

**Submission format**: You need to submit a zipped archive containing the jupyter notebook with your code and a pdf file named *project\_questions\_and\_solutions.pdf* (i.e., the current file with the solutions that you will fill in and convert to PDF format).

Each of your jupyter notebooks should be directly executable without any further modifications (i.e., if your assignment submission is downloaded from canvas and unzipped), any of your submitted notebooks if executed using a jupyter server should be directly runnable without additional data downloads etc.

Titanic Classification Problem

* 1. **Dataset Description:** The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. Our goal is to develop a classifier to predict whether a passenger *survived* the calamity. The attributes in the datasets are as follows:  
     1. PassengerID (categorical): Passenger ID
     2. Ticket (categorical): Ticket Number
     3. Name (categorical): Passenger Name
     4. Cabin (categorical): Passenger Cabin
     5. Pclass (categorical): Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
     6. Sex (categorical): Passenger Sex
     7. Age (continuous): Passenger Age
     8. SibSp (continuous): Number of Siblings/Spouses Aboard
     9. Parch (continuous): Number of Parents/Children Aboard
     10. Fare (continuous): Passenger Fare
     11. Embarked (categorical): Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
     12. Survived (categorical): Survival (0 = No;1 = Yes) --- this is the target column. Make sure to encode data appropriately for each classifier.
     13. Our target variable will be *Survived***.** Use the rest of the fields mentioned above to predict whether a passenger survived the Titanic shipwreck.
  2. **Data Loading / Preprocessing**
     1. Loading
        1. Load the data <df\_train.csv> and <df\_test.csv> as a pandas dataframe using the [‘pandas.read\_csv’](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html) function. The ‘df\_test.csv’ has been preprocessed (I.e., null values have been dropped, certain columns etc. have been dropped) and should not be changed apart from splitting the the dataframe into X\_test and y\_test. The ‘df\_train’ data has NOT been preprocessed and you will need to preprocess and prepare the ‘df\_train’ dataframe. Note: Neither *df*\_*train* nor *df\_test* have been scaled. The next few steps will enumerate data preprocessing, scaling requirements we need to perform.
        2. The resulting dataframe (i.e., *df\_train*) should have the shape (712,12) indicating that there are 712 instances and 12 columns.
        3. In *df\_train,* dataframe, currently you have 12 columns which are the following – PassengerID, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked and the Survived column (target variable).
        4. Use the [‘pandas.isnull](https://pandas.pydata.org/docs/reference/api/pandas.isnull.html)().sum()’ function check if there are any missing values in the df\_train dataframe. Report which columns have missing (i.e., null) values and provide the number of the null values in the columns.
        5. Use the [‘pandas.DataFrame.drop()’](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html) function to drop the ‘Cabin’, ‘PassengerID’, ‘Name’ and ‘Ticket’ columns.
        6. Use the [‘pandas.DataFrame.fillna()’](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html) function to replace the NA values in the ‘Age’ column with the mean value of the ‘Age’ column. Note: This process is called `imputation` (i.e., filling null values with a pre-specified value) and we are employing one strategy called *mean imputation,* but other strategies can also be employed in general. Use the [‘dropna()’](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html) function to drop any remaining rows that consist of NA values.
        7. Your task is to use the feature columns to predict the target column. This can be cast as a classification problem.
        8. Create a pandas dataframe X\_train of features (by dropping the ‘Survival’ column from the df\_train dataframe). Create a pandas *Series* object of targets y\_train (by only considering the ‘Survival’ column from the df\_train dataframe). Moving forward, we will be working with X\_train and y\_train. At this point also split the df\_test into X\_test and y\_test by dropping the ‘Survival’ column and storing the features into X\_test. Store the ‘Survival’ column in y\_test.
     2. Data Visualization
        1. Using matplotlib employ a scatter plot using [‘matplotlib.pyplot.scatter’](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html) between the age of the passengers and the price of their fare. Label the x-axis and the y-axis along with the giving the plot a title.
           1. What is the highest and lowest fare price?
           2. What are the respective mean values of these two features?
           3. What was the age of the oldest passenger?
        2. **Only for this question** use the df\_train dataframe. Using matplotlib visualize the number of males and females that survived and their respective *passenger classes* on two separate bar chart plots using [‘matplotlib.pyplot.bar’](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html) (Passenger Class column).
           1. Which *class of passengers* had the least number of survived males and how many? Repeat this analysis for females.
           2. Which *class of passengers* had the greatest number of survived males and how many? Repeat this for females.
        3. Using the Target variable (Survived) in y\_train plot a bar chart showing the distribution of the ‘Survived’ column.
           1. What initial comment can you make about this distribution in terms of the frequency of each class?
        4. So far you should have successfully been able to load, preprocess and visualize your data. Use the [‘pd.get\_dummies()’](https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html) function to convert categorical data into dummy variables (‘Sex’ and ‘Embarked’). Make sure to pass `drop\_first=True` to the `get\_dummies()` function. **(Perform this only on X\_train store the result back into X\_train).**
           1. What is the new shape of X?
     3. Data Scaling
        1. Employ X\_train and [MinMaxScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html) **only on the continuous attributes.** Employ the [‘fit\_transform()’](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html) function of the scaler to retrieve the new (scaled) version of the data. Store the scaled values in X\_train again.
        2. Scale the X\_test using the scaler you have just fit, this time using the [`transform()`](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html) function. Note: store the scaled values back into X\_test. At the end of this step, you must have X\_train, X\_test, all scaled according to the MinMaxScaler.
  3. **Modelling**
     1. Modelling (Model Instantiation / Training) using **Logistic Regression classifier**
        1. Employ the [Logistic Regression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) classifier from sklearn and instantiate the model. Label this model as ‘model\_lr’
        2. Once instantiated, `fit()` the model using the *scaled* X\_train, y\_train data.
        3. Employ the `predict()` function to obtain predictions on X\_test and store this in a variable labeled as ‘y\_pred\_lr’.
        4. Employ the [‘accuracy\_score’](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) function by using the ‘y\_pred\_lr’ and ‘y\_test’ variables as the functions parameters and print the accuracy of the Logistic Regression model.
     2. **Modelling** (Model Instantiation / Training) using **Support Vector Machine Classifier**
        1. Employ the [Support Vector Machine (SVM)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) classifier from sklearn () and instantiate the model. Make sure to set ‘probability=True’ when creating the model. Label this model as ‘model\_svm’
        2. Once instantiated, ‘fit()’ the model using the *scaled* X\_train, y\_train data.
        3. Employ the ‘predict()’ function to obtain predictions on X\_test and store this in a variable labeled as ‘y\_pred\_svm’.
        4. Employ the [‘accuracy\_score’](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) function by using the ‘y\_pred\_svm’ and ‘y\_test’ variables as the functions parameters and print the accuracy of the SVM model.and print the accuracy of the SVM model.
     3. **Modelling** **Logistic Regression Classifier** with the addition of noise on the target variable. In the data repository you should see three noisy datasets – ‘*df\_train\_noise20*’, ‘*df\_train\_40*’, ‘*df\_train\_60*’. These datasets have already been preprocessed. In each dataset `df\_train\_noise<integer>, the integer indicates the percentage of noise injected into the target variable in that training set. The noise can be considered a result of incorrect class labelling of a particular instance. For example, in df\_train\_noise20, 20% of the instances have an incorrect target label in the training set. Our goal will be to train a set of classification models on such noisy training data and test on a clean test set (i.e., same as what we have been using so far `df\_test’).   
        Load the 'df\_train\_noise<nl>' (nl means noise level and is a place holder for the integer percentage) datasets and split the data into X\_train\_<nl> and y\_train\_<nl> (e.g., if working with `df\_train\_noise20’ we would split the data and store it in variables named X\_train\_20, y\_train\_20). ‘X\_train\_<nl>’ should store the features and ‘y\_train\_<nl>’ should store the target variable.  
          
        Repeat the following steps (1 – 4) for the 20%, 40%, 60% noise level datasets.   
        Train a new Logistic Regression model on the new training and use the pre-existing X\_test and y\_test to evaluate your model. Label this model as ‘model\_lr\_noise\_<nl>’. Specifically, do the following:
        1. Employ a new Logistic Regression classifier from sklearn and instantiate the model. Label this model as ‘model\_lr\_noise<nl>’
        2. Once instantiated, ‘fit()’ the model using theX\_train\_<nl> and y\_train\_<nl> data.
        3. Employ the ‘predict()’ function to obtain predictions on X\_test and store this in a variable labeled as ‘y\_pred\_lr\_noise<nl>’.
        4. Employ the [‘accuracy\_score’](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) function and print the accuracy of the new Logistic Regression model.
           1. What is your initial observation of the accuracy, is the accuracy higher or lower than that of the clean dataset?
     4. **Modelling** **Support Vector Machine (SVM) Classifier** with the addition of noise on the target variable. In the data repository you should see three noisy datasets – ‘*df\_train\_noise20*’, ‘*df\_train\_40*’, ‘*df\_train\_60*’. These datasets have already been preprocessed. In each dataset `df\_train\_noise<integer>, the integer indicates the percentage of noise injected into the target variable in that training set. The noise can be considered a result of incorrect class labelling of a particular instance. For example, in df\_train\_noise20, 20% of the instances have an incorrect target label in the training set. Our goal will be to train a set of classification models on such noisy training data and test on a clean test set (i.e., same as what we have been using so far `df\_test’).   
        Load the 'df\_train\_noise<nl>' (nl means noise level and is a place holder for the integer percentage) datasets and split the data into X\_train\_<nl> and y\_train\_<nl> (e.g., if working with `df\_train\_noise20’ we would split the data and store it in variables named X\_train\_20, y\_train\_20). ‘X\_train\_<nl>’ should store the features and ‘y\_train\_<nl>’ should store the target variable.  
          
        Repeat the following steps (1 – 4) for the 20%, 40%, 60% noise datasets.   
        Train a new SVM classification model on the new training and use the pre-existing X\_test and y\_test to evaluate your model. Label this model as ‘model\_svm\_noise\_<nl>’. Specifically, do the following:
        1. Employ a new SVM classifier from sklearn and instantiate the model. Label this model as ‘model\_svm\_noise<nl>’
        2. Once instantiated, ‘fit()’ the model using theX\_train\_<nl> and y\_train\_<nl> data.
        3. Employ the ‘predict()’ function to obtain predictions on X\_test and store this in a variable labeled as ‘y\_pred\_svm\_noise<nl>’.
        4. Employ the [‘accuracy\_score’](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) function and print the accuracy of the new Logistic Regression model.
           1. What is your initial observation of the accuracy, is the accuracy higher or lower than that of the clean dataset?
  4. **Evaluation**
     1. Report F1 Score, Precision, Recall, Accuracy (All on the test set X\_test, y\_test)
        1. Employ a [`classification\_report()`](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html) function from sklearn.metrics to report the precision recall and f1 score for each class for the ‘model\_lr’ model and the ‘model\_svm’ along with a confusion matrix for each of them.
     2. Report the accuracy and classification report for each of the three noisy models (model\_lr\_noise<nl>, model\_svm\_noise<nl>).
     3. Make a calibration plot for each of the models without noisy data ('model\_lr', 'model\_svm'). Use the CalibrationDisplay class from sklearn.calibration to make your plot.
  5. **Discussion**
     1. Compare the performance of the Logistic Regression (model\_lr) classifier and the SVM classifier (model\_svm) trained on the clean training data sets. Using the classification reports which model performs better? Mention the specific numbers (i.e., from your results calculating the precision, recall, F1 score) in a table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model Name** | **Class** | **Precision** | **Recall** | **F1** |
| model\_lr | 0 |  |  |  |
| model\_lr | 1 |  |  |  |
| model\_svm | 0 |  |  |  |
| model\_svm | 1 |  |  |  |

* + 1. Report the performance of all the classification models you have trained thus far in terms of (precision, recall, F1 scores). From the results of the three noisy models implemented using Logistic Regression. What did you notice about the accuracy as the noise of the dataset increased? What can you say about the effect of noise on data mining pipelines?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model Name** | **Class** | **Precision** | **Recall** | **F1** |
| model\_lr | Not Survived |  |  |  |
| model\_lr | Survived |  |  |  |
| model\_lr\_noise20 | Not Survived |  |  |  |
| model\_lr\_noise20 | Survived |  |  |  |
| model\_lr\_noise40 | Not Survived |  |  |  |
| model\_lr\_noise40 | Survived |  |  |  |
| model\_lr\_noise60 | Not Survived |  |  |  |
| model\_lr\_noise60 | Survived |  |  |  |
| model\_svm | Not Survived |  |  |  |
| model\_svm | Survived |  |  |  |
| model\_svm\_noise20 | Not Survived |  |  |  |
| model\_svm\_noise20 | Survived |  |  |  |
| model\_svm\_noise40 | Not Survived |  |  |  |
| model\_svm\_noise40 | Survived |  |  |  |
| model\_svm\_noise60 | Not Survived |  |  |  |
| model\_svm\_noise60 | Survived |  |  |  |