DATA-445
Exam II Take-Home

Exam Instructions

1. **Show all your work** and write complete and coherent answers.
2. Show all of the steps that you used to get your final answer. If you do not show your work I can not give partial credit in the case of incorrect answers.
3. **Please strive for clarity and organization.**
4. **You are not allowed to discuss any of the exercises in Exam 2 take-home with others. Identical submissions will receive a 0 as a grade in Exam 1 take-home.**
5. **Late submission will not be accepted, regardless of the circumstances.**

Take the time to carefully read all the questions on the exam. GOOD LUCK!

1. Consider the `weather.csv` data file. This file contains weather data recorded in one minute interval from weather station located in San Diego, California. The weather station is equipped with sensors that capture weather-related measurements such as air temperature, air pressure, and relative humidity. Data was collected for a period of three years, from September 2011 to September 2014, to ensure that sufficient data for different seasons and weather conditions is captured. Each row in `weather.csv` contains weather data captured for a one-minute interval. Each row, or sample, consists of the following variables:

   - `rowID`: unique number for each row (Unit: NA)
   - `hpwren_timestamp`: timestamp of measure (Unit: year-month-day hour:minute:second)
   - `air_pressure`: air pressure measured at the timestamp (Unit: hectopascals)
   - `air_temp`: air temperature measure at the timestamp (Unit: degrees Fahrenheit)
   - `avg_wind_direction`: wind direction averaged over the minute before the timestamp (Unit: degrees, with 0 means coming from the North, and increasing clockwise)
   - `avg_wind_speed`: wind speed averaged over the minute before the timestamp (Unit: meters per second)
   - `max_wind_direction`: highest wind direction in the minute before the timestamp (Unit: degrees, with 0 being North and increasing clockwise)
   - `max_wind_speed`: highest wind speed in the minute before the timestamp (Unit: meters per second)
   - `min_wind_direction`: smallest wind direction in the minute before the timestamp (Unit: degrees, with 0 being North and inceasing clockwise)
   - `min_wind_speed`: smallest wind speed in the minute before the timestamp (Unit: meters per second)
   - `rain_accumulation`: amount of accumulated rain measured at the timestamp (Unit: millimeters)
   - `rain_duration`: length of time rain has fallen as measured at the timestamp (Unit: seconds)
   - `relative_humidity`: relative humidity measured at the timestamp (Unit: percent)

   In **Python**, answer the following:

   (a) (5 points) Using the `pandas` library, read the csv data file and create a data-frame called `weather`. Select data up to October 31, 2011. After that, remove any observation with missing values.

   (b) (8 points) After consulting with a meteorologist, he recommends to use the following variables for clustering purposes: `air_pressure`, `air_temp`, `avg_wind_direction`, `avg_wind_speed`, `max_wind_direction`, `max_wind_speed`, and `relative_humidity`. Transform all the variables of interest to 0-1 scale.

   (c) (8 points) Using the silhouette score, estimate the number of clusters for this dataset. Consider 2 to 20 clusters. Make sure to use `n_init = 20` in the `KMeans` function from the `sklearn.cluster` library.

(d) (6 points) Using the `KMeans` function from the `sklearn.cluster` library, cluster the customers into the number of clusters estimated from part (c).

(e) (8 points) Describe each of the clusters. Does the clustering results make sense? if not, suggest how would improve this analysis.

2. Consider the `churn-bigml-80.csv` and `churn-bigml-20.csv` datafile for this question. The Orange Telecom's churn dataset, which consists of cleaned customer activity data (features), along with a churn label specifying whether a customer canceled the subscription, will be used to develop predictive models. Each row represents a customer; each column contains customer's attributes. The datasets have the following attributes or features:

   - `State`: state where the customer live.
   - `Account_length`: number of months the account is active.
   - `Area_code`
   - `International_plan`: whether or not the customer has an international plan.
   - `Voice_mail_plan`: whether or not the customer has a voice mail plan.
   - `Number_vmail_messages`: number of voice mails.
   - `Total_day_minutes`
   - `Total_day_calls`
   - `Total_day_charge`
   - `Total_eve_minutes`
   - `Total_eve_calls`
   - `Total_eve_charge`
   - `Total_night_minutes`
   - `Total_night_calls`
   - `Total_night_charge`
   - `Total_intl_minutes`
   - `Total_intl_calls`
   - `Total_intl_charge`
   - `Customer_service_calls`
   - `Churn:` whether or not the customer churn.

   The goal is to build models that can help Orange Telecom to flag customers who likely to churn.

   (a) (5 points) Using the `pandas` library, read the csv data file and create two data-frames called: `telecom_train` (for `churn-bigml-80.csv`) and `telecom_test` (for `churn-bigml-20.csv`).

   (b) (12 points) Conduct the following feature engineering:

      - Using the `numpy` library, create a variable in `telecom_train` called `Churn_numb` that takes the value of 1 when `Churn = True` and 0 when `Churn = False`.

- Change the `International_plan` variable from a categorical variable to a numerical variable. That is, change `Yes` to 1 and `No` to 0 in both data-frames: `telecom_train` and `telecom_test`.
- Change the `Voice_mail_plan` variable from a categorical variable to a numerical variable. That is, change `Yes` to 1 and `No` to 0 in both data-frames: `telecom_train` and `telecom_test`.
- Create a new variable called: `total_charge` as the sum of `Total_day_charge`, `Total_eve_charge`, `Total_night_charge`, and `Total_intl_charge` in both data-frames: `telecom_train` and `telecom_test`.

(c) (5 points) In both data-frames `telecom_train` and `telecom_test`, only keep the following variables: `Account_length`, `International_plan`, `Voice_mail_plan`, `total_charge`, `Customer_service_calls`, and `Churn_numb`.

(d) (20 points) Consider the `telecom_train` dataset. Using `Account_length`, `International_plan`, `Voice_mail_plan`, `total_charge`, and `Customer_service_calls` as the input variables, and `Churn_numb` is the target variable. Do the following:

(1) Split the data into `train` (80%) and `test` (20%) taking into account the proportion of 0s and 1s in the data. That is, if Y is the target variable, in `train_test_split` function, you need to add the extra argument `stratify = Y`.

(2) Using the train dataset:

(i) Build a `RandomForestClassifier` model with `n_estimators = 500` and `max_depth = 3` on the train dataset. Extract the importance of variables.

(ii) Build a `ExtraTreesClassifier` model with `n_estimators = 500` and `max_depth = 3` on the train dataset. Extract the importance of variables.

(iii) Build a `GradientBoostingClassifier` model with `n_estimators = 500`, `max_depth = 3`, and `learning_rate = 0.01` on the train dataset. Extract the importance of variables.

Repeat steps (1)-(2) 100 times. Compute the average importance of each of the variables across the 1000 splits for the three models. After that, select the top 4 variables (the ones with top 4 average importance) as the predictor variables.

(e) (45 points) Consider the `telecom_train` dataset. Using `Churn_numb` as the target variable, and the top four variables from part (d) as the input variables. Do the following:

(i) Split the data into `train` (80%) and `test` (20%) taking into account the proportion of 0s and 1s in the data. That is, if Y is the target variable, in `train_test_split` function, you need to add the extra argument `stratify = Y`.

(ii) • Using the `train` dataset, build random forest models with the following setting: `n_tree = [100, 500, 1000, 1500, 2000]` and `depth = [3, 5, 7]`. In order to create a data-frame that contains all the combinations of trees and depths, you can use the following code:

```
import pandas as pd
from itertools import product

def expand_grid(dictionary):
    return pd.DataFrame([row for row in product(*dictionary.values())],
                         columns = dictionary.keys())

dictionary = {'n_tree': [100, 500, 1000, 1500, 2000],
              'depth': [3, 5, 7]}

parameters = expand_grid(dictionary)
```

For each random forest model that is built, use it to predict the likelihood of churn on the `test` dataset. Using 10% as the cut-off value, compute the accuracy and recall of each of the models.

- Using the `train` dataset, build extra trees models with the following setting: `n_tree = [100, 500, 1000, 1500, 2000]`, and `depth = [3, 5, 7]`. In order to create a data-frame that contains all the combinations of trees, depths, and learning rates, you can use the following code:

```
import pandas as pd
from itertools import product

def expand_grid(dictionary):
    return pd.DataFrame([row for row in product(*dictionary.values())],
                         columns = dictionary.keys())

dictionary = {'n_tree': [100, 500, 1000, 1500, 2000],
              'depth': [3, 5, 7]}

parameters = expand_grid(dictionary)
```

For each extra trees model that is built, use it to predict the likelihood of churn on the `test` dataset. Using 10% as the cut-off value, compute the accuracy and recall of each of the models.

- Using the `train` dataset, build gradient boosting models with the following setting: `n_tree = [100, 500, 1000, 1500, 2000]`, `depth = [3, 5, 7]`, and `learning_rate = [0.1, 0.01, 0.001]`. In order to create a data-frame that contains all the combinations of trees, depths, and learning rates, you can use the following code:

```
import pandas as pd
from itertools import product

def expand_grid(dictionary):
    return pd.DataFrame([row for row in product(*dictionary.values())],
                        columns = dictionary.keys())

dictionary = {'n_tree': [100, 500, 1000, 1500, 2000],
              'depth': [3, 5, 7],
              'learning_rate': [0.1, 0.01, 0.001]}

parameters = expand_grid(dictionary)
```

For each gradient boosting model that is built, use it to predict the likelihood of churn on the `test` dataset. Using 10% as the cut-off value, compute the accuracy and recall of each of the models.

(f) (30 points) Repeat part (e) 100 times. Identify the best model of each of the frameworks (based on the average accuracy and recall); that is, identify the best random forest model, the best extra-trees model, and the best gradient boosting model.

(g) (35 points) Using the `telecom_train` build three models: the best random forest model from part (f), the best extra-trees model form part (f), and the best gradient boosting model form part (f). Using these to three models, predict the likelihood of `Churn` on the `telecom_test` data-frame. After that, aggregate those likelihoods using the weighted average formula (use average recall of the models as weights). Using 10% as cutoff value, report the accuracy and recall of the aggregated predictions.

3. In this exercise, we will practice cross-validation and principal components. Let's consider the popular `digits dataset`. Notice that this will a multi-class classification task; that is, the goal is to predict the digit label: 0, 1, 2, 3, . . . , 8, 9.

   (a) (3 points) Load the `digits` dataset as follows:

   ```
   from sklearn.datasets import load_digits
   digits = load_digits()

   X = digits.data
   Y = digits.target
   ```

   (b) (30 points) Define the 10-folds cross-validation strategy with `RepeatedStratifiedKFold`. Make sure to set `n_repeats = 1` and set `random_state` equal to some positive integer (this argument is for reproducibility purposes).

      (i) Train a `LogisticRegression` (with default values) model over the 10-folds cross-validation strategy. Use `accuracy` to quantify model performance over the 10-folds. Make sure to use sta`cross_val_score` and put the input features on the same scale using `StandardScaler`. Report the average accuracy.

(ii) Train a `LinearDiscriminantAnalysis` model over the 10-folds cross-validation strategy. Use `accuracy` to quantify model performance over the 10-folds. Make sure to use `cross_val_score` and put the input features on the same scale using `StandardScaler`. Report the average accuracy.

(iii) Train a `RandomForestClassifier` (with default values) model over the 10-folds cross-validation strategy. Use `accuracy` to quantify model performance over the 10-folds. Make sure to use `cross_val_score`. Report the average accuracy.

(iv) Train a `ExtraTreesClassifier` (with default values) model over the 10-folds cross-validation strategy. Use `accuracy` to quantify model performance over the 10-folds. Make sure to use `cross_val_score`. Report the average accuracy.

(v) Train a `HistGradientBoostingClassifier` (with default values) model over the 10-folds cross-validation strategy. Use `accuracy` to quantify model performance over the 10-folds. Make sure to use `cross_val_score`. Report the average accuracy.

(vi) Train a `LGBMClassifier` (with default values) model over the 10-folds cross-validation strategy. Use `accuracy` to quantify model performance over the 10-folds. Make sure to use `cross_val_score`. Report the average accuracy.

(c) (30 points) From part (b), select the top 3 models in terms of accuracy. Then, do the following:

(i) Train the best model from part (b) over the 10-folds cross-validation strategy (defined in part (b)) using principal components that explained 95% of the variability of the data as the input features in the model. Make sure to standardize the input features with `StandardScaler` before you compute the principal components. Also, make sure to use `cross_val_score` and `Pipeline`. Report the average accuracy.

(ii) Train the second best model from part (b) over the 10-folds cross-validation strategy (defined in part (b)) using principal components that explained 95% of the variability of the data as the input features in the model. Make sure to standardize the input features with `StandardScaler` before you compute the principal components. Also, make sure to use `cross_val_score` and `Pipeline`. Report the average accuracy.

(iii) Train the third best model from part (b) over the 10-folds cross-validation strategy (defined in part (b)) using principal components that explained 95% of the variability of the data as the input features in the model. Make sure to standardize the input features with `StandardScaler` before you compute the principal components. Also, make sure to use `cross_val_score` and `Pipeline`. Report the average accuracy.

(iv) Using the results from parts (i)-(iii), what is the best model in terms of accuracy? Does this model outperform its previous version from part (b)? Why is this the reason behind difference in performance?