Contents

1 Basic Test Results

```
**********************************

HACKATHON PRESUBMISSION SCRIPT

**********************************

Extract your zip...
O.K.

Check all required files exist...
O.K.


You chose to submit Task 1! Cool!


Check USER.txt file format...
O.K.

Create a virtual env and install requirements...
-------------
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /tmp/tmp4zxs8hxb/task1/presubmission_env/bin/python3
Also creating executable in /tmp/tmp4zxs8hxb/task1/presubmission_env/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
Collecting numpy==1.16.3
  Downloading numpy-1.16.3-cp37-cp37m-manylinux1_x86_64.whl (17.3 MB)
Collecting sklearn==0.0
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Collecting scikit-learn
  Downloading scikit_learn-0.24.2-cp37-cp37m-manylinux2010_x86_64.whl (22.3 MB)
Collecting scipy>=0.19.1
  Downloading scipy-1.6.3-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
Collecting joblib>=0.11
  Downloading joblib-1.0.1-py3-none-any.whl (303 kB)
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)
Collecting scipy>=0.19.1
  Downloading scipy-1.6.2-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
  Downloading scipy-1.6.1-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
  Downloading scipy-1.6.0-cp37-cp37m-manylinux1_x86_64.whl (27.4 MB)
  Downloading scipy-1.5.4-cp37-cp37m-manylinux1_x86_64.whl (25.9 MB)
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py): started
  Building wheel for sklearn (setup.py): finished with status 'done'
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1309 sha256=be6ae25bc263bb978f9ee42a64c48eb1ab1a
  Stored in directory: /tmp/bodek.0x07F7/iml/test/ehudda/presubmission/home/.cache/pip/wheels/46/ef/c3/157e41f5ee1372d1be90b
Successfully built sklearn
Installing collected packages: numpy, threadpoolctl, scipy, joblib, scikit-learn, sklearn
Successfully installed joblib-1.0.1 numpy-1.16.3 scikit-learn-0.24.2 scipy-1.5.4 sklearn-0.0 threadpoolctl-2.1.0
-------------
O.K.

Test your code...
-------------
Collecting pandas
  Downloading pandas-1.2.4-cp37-cp37m-manylinux1_x86_64.whl (9.9 MB)
Collecting pytz>=2017.3
```

```
  Downloading pytz-2021.1-py2.py3-none-any.whl (510 kB)
Collecting python-dateutil>=2.7.3
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
Collecting numpy>=1.16.5
  Downloading numpy-1.20.3-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (15.3 MB)
Collecting six>=1.5
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, pytz, python-dateutil, numpy, pandas
  Attempting uninstall: numpy
    Found existing installation: numpy 1.16.3
    Uninstalling numpy-1.16.3:
      Successfully uninstalled numpy-1.16.3
Successfully installed numpy-1.20.3 pandas-1.2.4 python-dateutil-2.8.1 pytz-2021.1 six-1.16.0
Requirement already satisfied: numpy in ./presubmission_env/lib/python3.7/site-packages (1.20.3)
-------------

/tmp/tmp4zxs8hxb/task1/presubmission_env/lib/python3.7/site-packages/sklearn/base.py:315: UserWarning: Trying to unpickle es
  UserWarning)
Run your predict function using a tiny csv with 5 movies...
Index(['budget', 'vote_average', 'vote_count', 'runtime', 'status', 'revenue',
       'com_website', 'title_len', 'is_in_collection', 'en_language',
       'fr_language', 'hi_language', 'es_language', 'ja_language',
       'ru_language', 'it_language', 'ko_language', 'ta_language',
       'zh_language', 'genre_10749', 'genre_18', 'genre_35', 'genre_53',
       'genre_80', 'year', 'month'],
      dtype='object')

-----Results-----
Your revenues: [84342893.2271148, 202318958.452541, -14636219.685530901, -26787554.55711627, 1389178.9149868488]
Real revenues: [61621140, 94882889, 7527232, 34522221, 27105095]
RMSE for revenue: 58258394.41436296

Your avg votes: [6.671221167214171, 5.981693802267866, 6.251582841352626, 6.651459221385672, 6.082297970223522]
Real avg votes: [7.3, 6.5, 6.2, 7.4, 5.8]
RMSE for avg vote: 0.5112084439697623

O.K.



**********
YOU PASSED PRESUBMISSION!
**********
```

2 README.txt

```
1  ########################################
2  #                                      #
3  #           IML HACKATHON              #
4  #                                      #
5  ########################################
6
7
8  ############### Scripts ###############
9  regression.py
10 This file include 'predict' function as API demands
11
12
13 regression_class.py
14 - contain preproccing data
15 - all figures
16 - define the model
17 - save the model object as binary file
18
19 ############### Binary files ###############
20 our_model_revenue.bi - our model object
21 pop_words.bi - list of most popular words in overviews
```

3 USERS.txt

```
1  natan, 319192068
2  rachel, 208287334
3  nerya333, 313553299
4  ehudda, 20151275
```

IML HACKATHON

אהוד דהן, נתן גולדשטיין, תמר עשהאל, נריה כהן

IML_HACKATHON_2021

Aim We strive to create a model that, given data about movies, will successfully predict the revenue made in the box office and the average viewer ranking of those movies before they are officially released. The revenue is represented as an int, and the ranking is a float between 0 and 10, with one digit after the decimal point.

DATASET DESCRIPTION AND CHALLENGING CHARACTERISTICS:

Dataset description: The provided dataset contained around 5,000 movies, each with 22 features (ID, collection to which the movie belongs (if indeed), budget, genre, link to homepage, original language, original title, overview, number of viewers who ranked the movie, production companies, production countries, release date, runtime, spoken language, the stage of production, tagline, title, keywords, cast, crew.

Challenging Characteristics of the Dataset:

1. The multitude of formats that the different characteristics of the movies had. Some of the data (like the budget) was numeric, some was a long string (like the overview). Moreover, some columns included lists inside of them: the genres, crew, production company, keywords, cast and spoken languages, were all lists.
2. Nonnumeric data had to be processed before it could be used.
3. Missing data. Many values simply did not exist in the provided data. For example, many revenue values were missing.
4. Non-useful variables. Some of the features could be directly inferred from other features and were therefor pretty much useless in and of themselves.

DATA CLEANING AND PRE-PROCESSING

First,

we had to figure out a way to divide the provided data into a training set and a set for testing. We decided to do this by first sorting the all the data according to the revenue, and then picking every forth feature to be in the test set, the rest of the data went to the training set. We decided to fill missing fields using the average of the data set, thus we would have a better result than if we were simply to fill the missing space with a zero, and we do not change the balance of that feature too much. We dealt with nonnumeric data by using dummy variables. Now, we feel that since much of the course is in English, it is appropriate to write at least some of the document in Hebrew. So -

CONSIDERATIONS THAT GUIDED OUR DESIGN OF THE LEARNING SYSTEMS and DESCRIPTION OF THE CHOSEN ALGORITHM

VARIOUS METHODS WE TRIED AND THEIR RESULTS

PREDICTION (AND EXPLANATION) OF THE GENERAL MODEL ERROR WE EXPECT OUR SYSTEM TO HAVE:

5 task1/init our model.py

```
from parsing import *
from sklearn.linear_model import LinearRegression, Lasso
import pickle
from regression import predict
import plotly.graph_objects as go
import numpy as np
import plotly.express as px
import pandas as pd
from sklearn.linear_model import LinearRegression
import seaborn as sn
import matplotlib.pyplot as plt


def load_y(csv_file, colname):
    return pd.read_csv(csv_file)[colname]


def plot_rmse():
    # Plot rmse for increasing amount of samples
    results_lin = []
    # results_lasso = []
    X = load_data("../../Data/movies_dataset_part2.csv")
    y_test = X['revenue']
    y_test_vote = X['vote_average']
    X_test = X.drop['revenue', 'vote_average']
    for i in range(1, 101):
        linear_model = LinearRegression()
        # lasso_model = Lasso(alpha=1.0)
        n = max(round(X.shape[0] * (i / 100)), 1)

        X_test, y_test_rev, y_test_votes = basic_load_data("../Data/test_set.csv")
        results_lin.append(rmse(y_test_rev, linear_model.predict(X_test)))
        # results_lasso.append(rmse(y_test_rev, lasso_model.predict(X_test)))

    fig = go.Figure(go.Scatter(x=list(range(1, len(results_lin) + 1)), y=results_lin, mode="markers"),
                    layout=go.Layout(title="Model Evaluation Over Increasing Portions Of Training Set",
                                     xaxis=dict(title="Percentage of Training Set"),
                                     yaxis=dict(title="MSE Over Test Set")))
    fig.write_image("../Figures/mse.over.training.percentage.lin.png")

    fig = go.Figure(go.Scatter(x=list(range(1, len(results_lasso) + 1)), y=results_lin, mode="markers"),
                    layout=go.Layout(title="Model Evaluation Over Increasing Portions Of Training Set",
                                     xaxis=dict(title="Percentage of Training Set"),
                                     yaxis=dict(title="MSE Over Test Set")))
    fig.write_image("../Figures/mse.over.training.percentage.lasso.png")


def filter_by_corr_mat(X, y):
    """
    :param X: data frame
    :param y: vector
    :return: return list of choosen variables
    """
    mat = pd.concat([X, y], axis=1).corr()
    # print(mat.columns[mat['revenue'] > 0.2])


def split_data_by_zero_rev(filename):
    """
```

```
    Load movies prices dataset split the data
    :param filename: Path to movies prices dataset
    :return: Training_set = 3/4, Test_set = 1/4, with respect to the revenue field
    """
    df = pd.read_csv(filename).drop_duplicates()
    df0 = df[df['revenue'] == 0]
    df1 = df[df['revenue'] != 0]
    return df0, df1


def init_our_model():
    X = pd.read_csv("../../Data/Data_after_preproccecing.csv")
    y_rev = X['revenue']
    y_votes = X['vote_average']
    X = X.drop(['revenue', 'vote_average'], axis=1)

    model_list = [LinearRegression(), LinearRegression()]
    print("init X.shape", X.shape)
    print("init columns:", X.columns)
    model_list[0].fit(X, y_rev)
    model_list[1].fit(X, y_votes)

    outfile = open("our_models.bi", 'wb')
    pickle.dump(obj=model_list, file=outfile)
    outfile.close()


def cor_mat(X):
    df = pd.DataFrame(X)
    corrMatrix = df.corr()
    sn.heatmap(corrMatrix, annot=True)
    plt.show()
    plt.savefig("../../Figures/corr_mat.png")


def rmse(y, y_pred):
    """
    Calculate the MSE given the true- and prediction- vectors
    :param y: The true response vector
    :param y_pred: The predicted response vector
    :return: MSE of the prediction
    """
    return np.sqrt(np.mean((y - y_pred) ** 2))


def plot_singular_values(X):
    """
    Given a design matrix X, plot the singular values of all non-categorical features
    :param X: The design matrix to use
    """
    sv = np.linalg.svd(X, compute_uv=False)
    fig = go.Figure(go.Scatter(x=X.columns, y=sv, mode='lines+markers'),
                    layout=go.Layout(title="Scree Plot of Design Matrix Singular Values",
                                     xaxis=dict(title=""), yaxis=dict(title="Singular Values")))
    fig.write_image("../Figures/singular.values.scree.plot.png")


def feature_evaluation(X, y):
    for f in X:
        rho = np.cov(X[f], y)[0, 1] / (np.std(X[f]) * np.std(y))

        fig = px.scatter(pd.DataFrame({'x': X[f], 'y': y}), x="x", y="y", trendline="ols",
                         title=f"Correlation Between {f} Values and Response <br>Pearson Correlation {rho}",
                         labels={"x": f"{f} Values", "y": "Response Values"})
        fig.write_image("../Figures/pearson.correlation.%s.png" % f)
```

```
128
129 if __name__ == '__main__':
130     # load_data("../../Data/movies_dataset.csv", True)
131     # cor_mat(pd.read_csv("../../Data/Data_after_preproccecing.csv"))
132     # plot_rmse()
133     init_our_model()
```

6 task1/model columns.csv

```
budget,vote_average,vote_count,runtime,status,revenue,com_website,title_len,is_in_collection,en_language,fr_language,hi_lang
2000000,7.2,304,105.0,1,21665468,0,24,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,1992,4
```

7 task1/parsing.py

```python
import pandas as pd
from ast import literal_eval
import pickle
import re

LANGUAGE_NAMES = ['en', 'fr', 'hi', 'es', 'ja', 'ru', 'it', 'ko', 'ta', 'zh']
POP_GENRES = ["genre_10749", "genre_18", "genre_35", "genre_53", "genre_80", "genre_18"]
POP_CREW = ["crew_4185"]
COLS_DROP = ["id", "belongs_to_collection", "genres", "homepage", "original_language", "original_title", "overview",
             "production_companies", "production_countries", "release_date", "spoken_languages", "tagline", "title",
             "keywords", "cast", "crew"]
COLS_DROP2 = ["id_genres",
              "id_production_companies", "id_keywords", "id_countries",
              "id_lan", "pop_words", "id_collection"]
id_list = ['genres', 'production_companies', 'keywords']
iso_list = ['production_countries', 'spoken_languages']
crew_list = ['cast', 'crew']

pd.options.mode.chained_assignment = None


def literal_converter_id(val):
    # replace first val with '' or some other null identifier if required
    return {'id': pd.NA} if (val == "") or (val == "[]") else literal_eval(val)


def literal_converter_iso(val):
    return {'iso_3166_1': pd.NA} if (val == "") or (val == "[]") else literal_eval(val)


def literal_converter_lan(val):
    return {'iso_639_1': pd.NA} if (val == "") or (val == "[]") else literal_eval(val)


def literal_converter_crew(val):
    return {'id': pd.NA} if (val == "") or (val == "[]") else literal_eval(val)


def time_variable(df):
    """get a df with release_date time-variable as 16/04/1992 and split to years and 12 months"""
    df['release_date'] = pd.to_datetime(df['release_date'])
    df['year'] = df['release_date'].dt.year
    df['month'] = df['release_date'].dt.month
    return df


def create_pop_words_list(words_set):
    """words_set - lists of words
    count number of known words in movie description
    Known word are the 10% percent of incident words in the data set
    @return list of first 10% highest words in the data set"""
    # print(words_set[:51])
    words_amounts = dict()
    for i in range(len(words_set)):
        try:
            # print("i=", i, ":", words_set[i])
            sentence = words_set[i].split(' ')
            for j in range(len(sentence)):
                word = sentence[j].lower()
```

```
                if word in words_amounts:
                    words_amounts[word] += 1
                else:
                    words_amounts[word] = 1
        except:
            pass
    pop_words = sorted(words_amounts, key=words_amounts.get, reverse=True)[:1000]
    outfile = open("Data/pop_words.bi", 'wb')
    pickle.dump(obj=pop_words, file=outfile)
    outfile.close()


def words_dict(words_set):
    infile = open("pop_words.bi", 'rb')
    pop_words = pickle.load(infile)
    infile.close()

    res = []
    for i in range(len(words_set)):
        l = []
        try:
            sentence = words_set[i].split(' ')
            for j in range(len(sentence)):
                word = sentence[j].lower()
                if word in pop_words:
                    l.append(word)
        except:
            pass
        res.append(l)
    return res


def parse_jsons(df):
    '''
    parse json column to relevant columns

    '''
    df['id_collection'] = pd.DataFrame([list(pd.json_normalize(c)['id'].values) for c in df['belongs_to_collection']])
    id_list = ['genres', 'production_companies', 'keywords']
    for colname in id_list:
        df['id_' + str(colname)] = [str(pd.json_normalize(c)['id'].tolist())[1:-1] for c in df[colname]]
    df['id_countries'] = [str(pd.json_normalize(c)['iso_3166_1'].to_list())[1:-1] for c in df['production_countries']]
    df['id_lan'] = [str(pd.json_normalize(c)['iso_639_1'].to_list())[1:-1] for c in df['spoken_languages']]
    # df['cast_ids'] = [str(pd.json_normalize(c)['id'].to_list())[1:-1] for c in df['cast']]
    # df['crew_ids'] = [str(pd.json_normalize(c)['id'].to_list())[1:-1] for c in df['crew']]

    df['com_website'] = df.homepage.apply(lambda x: 1 if re.match(r".com.", str(x)) else 0)
    df['title_len'] = df.original_title.apply(lambda x: len(str(x)))
    # create_pop_words_list(df.overview) # Need to run only once
    df['pop_words'] = words_dict(df.overview)
    return df


def add_dummies(df):
    df['status'] = df['status'].apply(lambda x: 1 if x == "Released" else 0)
    df['is_in_collection'] = df['id_collection'].notna().astype('int')
    for l in LANGUAGE_NAMES:
        df[l + "_language"] = (df["original_language"] == l).astype('int')
    temp = df['id_genres'].str.get_dummies(sep=',').rename(lambda x: 'genre_' + x, axis='columns')
    for g in POP_GENRES:
        if g in temp.columns:
            df[g] = temp[g]
        else:
            df[g] = 0

    # temp_crew = df['crew_ids'].str.get_dummies(sep=',').rename(lambda x: 'crew_' + x, axis='columns')
    # temp_cast = df['cast_ids'].str.get_dummies(sep=',').rename(lambda x: 'cast_' + x, axis='columns')
    # df = pd.concat(df, temp_crew)
```

```
128         # for c in POP_CREW:
129         #     if c in temp_crew.columns:
130         #         df[c] = temp_crew[c]
131         #     else:
132         #         df[c] = 0
133 
134         # df = pd.concat([df, df['status'].str.get_dummies(sep=',').rename(lambda x: 'status_' + x, axis='columns')], axis=1)
135         # .drop(["status_<NA>"], axis=1)
136         return df
137 
138 
139     def fill_missing(df, avg):
140         for col in avg:
141             df[col].replace(to_replace=0, value=avg[col])
142 
143 
144     def find_avg(df):
145         """ find avg or most common item in column or """
146         avg = dict()
147 
148         l = ['budget', 'vote_count', 'runtime', 'month', 'year']
149 
150         for i in l:
151             avg[i] = ((df[df[i] != 0])[i].mean(skipna=True))
152             if i != 'vote_average':
153                 avg[i] = int(avg[i])
154 
155         outfile = open("avg_dict.bi", 'wb')
156         pickle.dump(obj=avg, file=outfile)
157         outfile.close()
158 
159 
160     def filter_training_data(X):
161         """"
162         filter not relevant columns for training
163         """
164         X = X[X['revenue'] != 0]
165         X = X[X['budget'] != 0]
166         return X
167 
168 
169     def zero_nan_carring(df, write_dict):
170         """
171         load the avg dictionaty and fill the values
172         :param df:
173         :return: df
174         """
175         if write_dict:
176             find_avg(df)
177         infile = open("avg_dict.bi", 'rb')
178         avg_dict = pickle.load(infile)
179         infile.close()
180 
181         df = df.fillna(0)
182 
183         fill_missing(df, avg_dict)
184         return df
185 
186 
187     def add_missing_columns(df):
188         """
189         add missing columns to the data
190         :param df:
191         :return: df
192         """
193         column_list = pd.read_csv("model_columns.csv").columns
194         print(column_list)
195         for col in column_list:
```

```
196             if col not in df.columns:
197                 df[col] = 0
198         return df
199 
200 
201     def normalized(df, write_dict):
202         """
203         Normalise and standartized variables
204         :param df:
205         :return:
206         """
207         if write_dict:
208             std_dict = {}
209             for col in df.columns:
210                 if max(df[col]) > 10:
211                     std_dict[col] = df[col].mean(), df[col].std()
212                     mu, std = std_dict[col]
213                     df[col] = (df[col] - mu) / std
214             outfile = open("std_dict.bi", 'wb')
215             pickle.dump(obj=std_dict, file=outfile)
216             outfile.close()
217         else:
218             infile = open("std_dict.bi", 'rb')
219             std_dict = pickle.load(infile)
220             infile.close()
221             for col in std_dict.keys():
222                 mu, std = std_dict[col]
223                 df[col] = (df[col] - mu) / std
224         return df
225 
226 
227 
228 
229     def load_data(csv_file, train_run=False):
230         """
231         get csv file path
232         make all preprocessing
233         return pandas data frame
234         :param csv_file:
235         :return:
236         """
237         df = pd.read_csv(csv_file, converters={'belongs_to_collection': literal_converter_id,
238                                                'genres': literal_converter_id,
239                                                'production_companies': literal_converter_id,
240                                                'keywords': literal_converter_id,
241                                                'production_countries': literal_converter_iso,
242                                                'spoken_languages': literal_converter_lan,
243                                                'cast': literal_converter_crew,
244                                                'crew': literal_converter_crew
245                                                })
246         if train_run:
247             df = filter_training_data(df)
248         df = parse_jsons(df)
249         df = add_dummies(df)
250         df = time_variable(df)
251         df = zero_nan_carring(df, train_run)
252 
253         df = df.drop(COLS_DROP + COLS_DROP2, axis=1)
254         # df = normalized(df, train_run)
255 
256         if train_run:
257             df.to_csv("../../Data/Data_after_preproccecing.csv", index=False)
258             df[:1].to_csv("model_columns.csv", index=False)
259         else:
260             df = add_missing_columns(df)
261             if 'revenue' in df.columns:
262                 df = df.drop(['revenue'], axis=1)
263             if 'vote_average' in df.columns:
```

```
264                 df = df.drop(['vote_average'], axis=1)
265
266         return df
```

8 task1/regression.py

```
1  ################################################
2  #
3  #     Task 1 - predict movies revenue & ranking
4  #
5  ################################################
6
7  from parsing import *
8  import pickle
9
10
11 def predict(csv_file):
12     """
13     This function predicts revenues and votes of movies given a csv file with movie details.
14     Note: Here you should also load your model since we are not going to run the training process.
15     :param csv_file: csv with movies details. Same format as the training dataset csv.
16     :return: a tuple - (a python list with the movies revenues, a python list with the movies avg_votes)
17     """
18     X = load_data(csv_file)
19     infile = open("our_models.bi", 'rb')
20     models_list = pickle.load(infile)
21     infile.close()
22     return list(models_list[0].predict(X)), list(models_list[1].predict(X))
23
24
25 if __name__ == '__main__':
26     predict("../../Data/movies_dataset_part2_test.csv")
```

9 task1/requirements.txt

```
1 numpy==1.16.3
2 sklearn==0.0
```