Machine Learning for Beer

Your goal is to train a model to predict the bitterness of a beer (in International Bittering Units, or IBU), given features about the beer. You can acquire the data in any one of three places:

- on Kaggle
- on Github (https://dlsun.github.io/pods/data/beer/beer_train.csv and https://dlsun.github.io/pods/data/beer/beer_test.csv)

A description of the variables is available here.

```
import pandas as pd
df_beer = pd.read_csv("https://dlsun.github.io/pods/data/beer/beer_train.csv")
df_beer
```

	id	abv	available	description	glass	ibu	isOrganic	name	originalGra
0	0	8.2	Available at the same time of year, every year.	A Belgian-Abbey-Style Tripel that is big in al...	NaN	31.0	N	LoonyToonTripel	
1	1	5.7	Available at the same time of year, every year.	Covert Hops is a crafty ale. Its stealthy dark...	Pint	45.0	N	Covert Hops	
2	2	5.8	Available at the same time of year, every year.	This is a traditional German-style Marzen char...	Mug	25.0	N	Oktoberfest	
3	3	5.5	Available year round as a staple beer.	A West Coast-Style Pale Ale balancing plenty o...	Pint	55.0	N	Pale Ale	
4	4	4.8	Available year round as a staple beer.	This Bombshell has a tantalizing crisp and cle...	Pint	11.4	N	Head Turner Blonde Ale	
...	...	...	...	...	...	...	...	...	
5995	5995	5.5	Available year round as a staple beer.	Taking its cues from “Three Threads”, a barten...	Pint	33.0	N	Mayflower Porter	
5996	5996	11.0	Available at the same time of year, every year.	Our barley wine is what would be considered an...	NaN	30.0	N	Barbieswine	

Question 1 (4 points)

You would like to predict **ibu** using a 20-nearest neighbors model. You are choosing between 4 sets of features to put into this model:

1. **abv**
2. **abv**, **name**
3. **abv**, **name**, **available**
4. **abv**, **name**, **available**, **glass**

Apply TF-IDF (using the top 100 terms) to the raw text variables and one-hot encoding to the categorical variables.

For each set of features, train a 20-nearest neighbor model to predict IBU (**ibu**). Which of these models is best for predicting IBU? Justify your answer.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

pipeline_abv = make_pipeline(
    make_column_transformer(
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
      ),
      KNeighborsRegressor(n_neighbors=20, metric="euclidean"))

pipeline_name = make_pipeline(
    make_column_transformer(
      (TfidfVectorizer(max_features = 100), "name"),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    KNeighborsRegressor(n_neighbors=20, metric="euclidean"))

pipeline_no_glass = make_pipeline(
    make_column_transformer(
```

```
      (TfidfVectorizer(max_features = 100), "name"),
      (OneHotEncoder(sparse=False, handle_unknown="ignore"), ["available"]),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    KNeighborsRegressor(n_neighbors=20, metric="euclidean"))

pipeline_all = make_pipeline(
    make_column_transformer(
      (TfidfVectorizer(max_features = 100), "name"),
      (OneHotEncoder(sparse=False, handle_unknown="ignore"), ["available", "glass"]),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    KNeighborsRegressor(n_neighbors=20, metric="euclidean"))

test_features = [["abv"],
                 ["abv", "name"],
                 ["abv", "name", "available"],
                 ["abv", "name", "available", "glass"]]

pipelines = [pipeline_abv, pipeline_name, pipeline_no_glass, pipeline_all]

for i in range(4):
  print(test_features[i], -cross_val_score(
                    pipelines[i], X_train, y_train,
                    scoring="neg_mean_squared_error", cv=10).mean())
```

```
['abv'] 650.3440049391983
['abv', 'name'] 554.8118078381209
['abv', 'name', 'available'] 575.2226271908737
['abv', 'name', 'available', 'glass'] 579.1486281896525
```

Features ["abv", "name"] are best at predicting ibu as seen by the lowest mean squared error after applying cross val score.

Question 2 (4 points)

Let's see how the distance metric and the scaling method influence prediction accuracy. Use the set of features from Question 1 that you determined to be the best. Continue to use $k = 20$ nearest neighbors, but try fitting models with different distance metrics and scaling methods. Which distance metric and/or scaling method gives the best prediction accuracy?

```
print(pipeline_name.get_params().keys())
```

```
dict_keys(['memory', 'steps', 'verbose', 'columntransformer', 'kneighborsregressor', 'columntransformer__n_jobs', 'columntransformer__rema
```

```
print(pipeline_name.get_params())
```

```
{'memory': None, 'steps': [('columntransformer', ColumnTransformer(transformers=[('tfidfvectorizer',
                                    TfidfVectorizer(max_features=100), 'name'),
                                   ('standardscaler', StandardScaler(), ['abv'])])), ('kneighborsregressor', KNeighborsRegressor(metric='eucl
                                    TfidfVectorizer(max_features=100), 'name'),
                                   ('standardscaler', StandardScaler(), ['abv'])]), 'kneighborsregressor': KNeighborsRegressor(metric='euclid
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler, RobustScaler
X_train, y_train = (df_beer[["abv", "name"]],
                    df_beer["ibu"])
grid_cv = GridSearchCV(
    pipeline_name,
    param_grid={
        "columntransformer": [ColumnTransformer(transformers=[('tfidfvectorizer',
                               TfidfVectorizer(max_features=100), 'name'),
                              ('standardscaler', StandardScaler(), ['abv'])]),
                             ColumnTransformer(transformers=[('tfidfvectorizer',
                               TfidfVectorizer(max_features=100), 'name'),
                              ('minmaxscaler', MinMaxScaler(), ['abv'])]),
                             ColumnTransformer(transformers=[('tfidfvectorizer',
                               TfidfVectorizer(max_features=100), 'name'),
                              ('robustscaler', RobustScaler(), ['abv'])])],
       "kneighborsregressor__metric": ["euclidean", "manhattan", "cosine"],
    },
    scoring="neg_mean_squared_error", cv=10)

grid_cv.fit(X_train, y_train)
grid_cv.best_params_
```

```
{'columntransformer': ColumnTransformer(transformers=[('tfidfvectorizer',
                                    TfidfVectorizer(max_features=100), 'name'),
                                   ('robustscaler', RobustScaler(), ['abv'])]),
 'kneighborsregressor__metric': 'manhattan'}
```

Robust Scaler and Manhattan distance metric gives best prediction accuracy.

Question 3 (4 points)

Finally, let's determine the right value of k. Use the set of features, the distance metric, and the scaling method that you determined to be best (for $k = 20$) in Questions 1 and 2. Fit k-nearest neighbor models for different values of k. Plot the training error and the estimated test error as functions of k, and determine the optimal value of k.

```
pipeline_best = make_pipeline(
    make_column_transformer(
        (TfidfVectorizer(max_features=100), 'name'),
        (RobustScaler(), ['abv'])
    ),
    KNeighborsRegressor(n_neighbors=20, metric="manhattan"))


X_train, y_train = (df_beer[["abv", "name"]],
                    df_beer["ibu"])
grid_cv = GridSearchCV(
     pipeline_best,
     param_grid={
         "kneighborsregressor__n_neighbors": range(1, 51)
     },
     scoring="neg_mean_squared_error", cv=10)

grid_cv.fit(X_train, y_train)
grid_cv.best_params_
```

```
    {'kneighborsregressor__n_neighbors': 33}
```

33 is the optimal value of k.

+ Code + Text

Question 4 (10 points)

The goal of machine learning is to build models with high predictive accuracy. Thus, it is not surprising that there exist machine learning competitions, where participants compete to build the model with the lowest possible prediction error.

Kaggle is a website that hosts machine learning competitions. In this lab, you will participate in a Kaggle competition with other students in this class! The top 5 people will earn up to 5 bonus points on this lab. The winners will earn additional (non-academic) prizes. To join the competition, visit this link. You will need to log into Kaggle, but you can use your Google account.

Train many different models to predict IBU. Try different subsets of variables. Try different machine learning algorithms (you are not restricted to just k-nearest neighbors). At least one of your models must contain variables derived from the `description` of each beer. Use cross-validation to systematically select good models and submit your predictions to Kaggle. You are allowed 2 submissions per day, so submit early and often!

Note that to submit your predictions to Kaggle, you will need to export your predictions to a CSV file (using `.to_csv()`) in the format expected by Kaggle (see `beer_test_sample_submission.csv` for an example).

You must upload at least 2 submissions to Kaggle to earn full credit.

```
pipeline_knn = make_pipeline(
    make_column_transformer(
        (TfidfVectorizer(max_features=100), 'name'),
        (RobustScaler(), ['abv'])
    ),
    KNeighborsRegressor(n_neighbors=33, metric="manhattan"))

df_test = pd.read_csv("https://dlsun.github.io/pods/data/beer/beer_test.csv")
X_test, y_test = df_test[["abv", "name"]], df_test["ibu"]

pipeline_knn.fit(X_train, y_train)
y_train_ = pipeline_knn.predict(X_test)

df_test["ibu"] = y_train_
df_test
```

	id	abv	available	description	glass	ibu	isOrganic	name	ori
0	6000	10.0	Limited availability.	A classic Belgian Trappist style strong ale wi...	Tulip	63.136364	N	She WILL!	
1	6001	5.2	Available year round as a staple beer.	An American-style of Pale Ale brewed with a ba...	Pint	36.551515	N	Defender American Pale Ale	
2	6002	4.0	Available during the winter months.	This amber wheat ale has a balanced malt body,...	Tulip	23.045455	Y	Hazel	
3	6003	10.2	Available year round as a staple beer.	A uniquely large beer developed by taking our ...	Pint	87.545455	N	Cinderella's Twin Double IPA	
4	6004	6.0	Limited availability.	An American red ale with crisp hop flavor.	NaN	32.500000	N	Independence Ale	

```
from google.colab import files
df_test[["id", "ibu"]].to_csv('output_knn.csv', encoding = 'utf-8-sig', index=False)
files.download('output_knn.csv')
```

Don't dismiss Little

```
from sklearn.linear_model import LinearRegression
pipeline_linear = make_pipeline(
    make_column_transformer(
      (TfidfVectorizer(max_features = 100), "name"),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    LinearRegression())

name_features = ["abv", "name"]
X_train, y_train = (df_beer[name_features],
                    df_beer["ibu"])
print("pipeline_linear", -cross_val_score(
                    pipeline_linear, X_train, y_train,
                    scoring="neg_mean_squared_error", cv=10).mean())

pipeline_linear.fit(X_train, y_train)
y_train_ = pipeline_linear.predict(X_test)
df_test["ibu"] = y_train_
df_test["ibu"]

df_linear = df_test[["id", "ibu"]]
df_linear.to_csv("output_linear.csv", index=False)
files.download('output_linear.csv')
```

```
pipeline_linear 524.6125563383266
```

```
from sklearn.tree import DecisionTreeRegressor
pipeline_dec = make_pipeline(
    make_column_transformer(
      (TfidfVectorizer(max_features = 100), "name"),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    DecisionTreeRegressor())

name_features = ["abv", "name"]
X_train, y_train = (df_beer[name_features],
                    df_beer["ibu"])
print("pipeline_dec", -cross_val_score(
                    pipeline_dec, X_train, y_train,
                    scoring="neg_mean_squared_error", cv=10).mean())
pipeline_dec.fit(X_train, y_train)
y_train_ = pipeline_dec.predict(X_test)
df_test["ibu"] = y_train_
df_test["ibu"]

df_dec = df_test[["id", "ibu"]]
df_dec.to_csv("output_dec.csv", index=False)
files.download('output_dec.csv')
```

```
pipeline_dec 1080.832059320547
```

```
from sklearn.ensemble import RandomForestRegressor

pipeline_rfr = make_pipeline(
    make_column_transformer(
      (TfidfVectorizer(max_features = 100), "name"),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    RandomForestRegressor())

    pipeline_rfr 764.2079206933374


pipeline_rfr.get_params()

    {'memory': None,
     'steps': [('columntransformer',
       ColumnTransformer(transformers=[('tfidfvectorizer',
                                        TfidfVectorizer(max_features=100), 'name'),
                                       ('standardscaler', StandardScaler(), ['abv'])])),
      ('randomforestregressor', RandomForestRegressor())],
     'verbose': False,
     'columntransformer': ColumnTransformer(transformers=[('tfidfvectorizer',
                                    TfidfVectorizer(max_features=100), 'name'),
                                   ('standardscaler', StandardScaler(), ['abv'])]),
     'randomforestregressor': RandomForestRegressor(),
     'columntransformer__n_jobs': None,
     'columntransformer__remainder': 'drop',
     'columntransformer__sparse_threshold': 0.3,
     'columntransformer__transformer_weights': None,
     'columntransformer__transformers': [('tfidfvectorizer',
       TfidfVectorizer(max_features=100),
       'name'),
      ('standardscaler', StandardScaler(), ['abv'])],
     'columntransformer__verbose': False,
     'columntransformer__verbose_feature_names_out': True,
     'columntransformer__tfidfvectorizer': TfidfVectorizer(max_features=100),
     'columntransformer__standardscaler': StandardScaler(),
     'columntransformer__tfidfvectorizer__analyzer': 'word',
     'columntransformer__tfidfvectorizer__binary': False,
     'columntransformer__tfidfvectorizer__decode_error': 'strict',
     'columntransformer__tfidfvectorizer__dtype': numpy.float64,
     'columntransformer__tfidfvectorizer__encoding': 'utf-8',
     'columntransformer__tfidfvectorizer__input': 'content',
     'columntransformer__tfidfvectorizer__lowercase': True,
     'columntransformer__tfidfvectorizer__max_df': 1.0,
     'columntransformer__tfidfvectorizer__max_features': 100,
     'columntransformer__tfidfvectorizer__min_df': 1,
     'columntransformer__tfidfvectorizer__ngram_range': (1, 1),
     'columntransformer__tfidfvectorizer__norm': 'l2',
     'columntransformer__tfidfvectorizer__preprocessor': None,
     'columntransformer__tfidfvectorizer__smooth_idf': True,
     'columntransformer__tfidfvectorizer__stop_words': None,
     'columntransformer__tfidfvectorizer__strip_accents': None,
     'columntransformer__tfidfvectorizer__sublinear_tf': False,
     'columntransformer__tfidfvectorizer__token_pattern': '(?u)\\b\\w\\w+\\b',
     'columntransformer__tfidfvectorizer__tokenizer': None,
     'columntransformer__tfidfvectorizer__use_idf': True,
     'columntransformer__tfidfvectorizer__vocabulary': None,
     'columntransformer__standardscaler__copy': True,
     'columntransformer__standardscaler__with_mean': True,
     'columntransformer__standardscaler__with_std': True,
     'randomforestregressor__bootstrap': True,
     'randomforestregressor__ccp_alpha': 0.0,
     'randomforestregressor__criterion': 'squared_error',
     'randomforestregressor__max_depth': None,
     'randomforestregressor__max_features': 'auto',
     'randomforestregressor__max_leaf_nodes': None,
     'randomforestregressor__max_samples': None,
     'randomforestregressor__min_impurity_decrease': 0.0,
     'randomforestregressor__min_samples_leaf': 1,
     'randomforestregressor__min_samples_split': 2,
     'randomforestregressor__min_weight_fraction_leaf': 0.0,


grid_cv = GridSearchCV(
     pipeline_rfr,
     param_grid={
        "randomforestregressor__max_depth" : range(1, 6),
        "randomforestregressor__min_samples_leaf" : range(1, 10),
     },
     scoring="neg_mean_squared_error", cv=10)

grid_cv.fit(X_train, y_train)
grid_cv.best_params_

    {'randomforestregressor__max_depth': 5,
     'randomforestregressor__min_samples_leaf': 8}
```

```
pipeline_rfr = make_pipeline(
    make_column_transformer(
      (TfidfVectorizer(max_features = 100), "name"),
      (StandardScaler(), ["abv"]),
      remainder = 'drop'
    ),
    RandomForestRegressor(max_depth = 5, min_samples_leaf = 8))

X_train, y_train = df_beer[["abv", "name"]], df_beer["ibu"]

pipeline_rfr.fit(X_train, y_train)
y_train_ = pipeline_rfr.predict(X_test)
df_test["ibu"] = y_train_
df_test["ibu"]

df_rfr = df_test[["id", "ibu"]]
df_rfr.to_csv("output_rfr.csv", index=False)
files.download('output_rfr.csv')
```

Submission Instructions

- Restart this notebook and run the cells from beginning to end.
    - Go to Runtime > Restart and Run All.
- Save this notebook as a PDF.
    - Go to File > Print and save as PDF.
    - Try printing with margins set to "None".
- Double check that all of your code and output is visible in the PDF.
- When you are done, upload the PDF to Gradescope.

The assignment is due Monday, February 20 at 11:59 PM.