5B. Prediction Competiton

February 26, 2019

1 Prediction Competition

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. To join the competition, visit this link. You will need to create an account on Kaggle first.

2 Question

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).

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn.feature_extraction import DictVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import QuantileTransformer
```

```
beer = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/master/da
beer.glass = beer.glass.fillna("None")
beer.head()
```

Out[1]:
```
   id  abv                                        available  \
0   0  8.2  Available at the same time of year, every year.
1   1  5.7  Available at the same time of year, every year.
2   2  5.8  Available at the same time of year, every year.
3   3  5.5            Available year round as a staple beer.
4   4  4.8            Available year round as a staple beer.

                                         description glass   ibu isOrganic  \
0  A Belgian-Abbey-Style Tripel that is big in al...  None  31.0         N
1  Covert Hops is a crafty ale. Its stealthy dark...  Pint  45.0         N
2  This is a traditional German-style Marzen char...   Mug  25.0         N
3  A West Coast-Style Pale Ale balancing plenty o...  Pint  55.0         N
4  This Bombshell has a tantalizing crisp and cle...  Pint  11.4         N

                     name  originalGravity srm
0         LoonyToonTripel            1.070   8
1             Covert Hops            1.056  35
2             Oktoberfest            1.048  10
3                Pale Ale            1.044   5
4  Head Turner Blonde Ale            1.045   3
```

In [2]:
```
beer_test = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/mast
beer_test.glass = beer.glass.fillna("None")
beer_test.head()
```

Out[2]:
```
     id   abv                               available  \
0  6000  10.0                    Limited availability.
1  6001   5.2  Available year round as a staple beer.
2  6002   4.0     Available during the winter months.
3  6003  10.2  Available year round as a staple beer.
4  6004   6.0                    Limited availability.

                                         description glass  ibu isOrganic  \
0  A classic Belgian Trappist style strong ale wi...  None  NaN         N
1  An American-style of Pale Ale brewed with a ba...  Pint  NaN         N
2  This amber wheat ale has a balanced malt body,...   Mug  NaN         Y
3  A uniquely large beer developed by taking our ...  Pint  NaN         N
4         An American red ale with crisp hop flavor.  Pint  NaN         N

                          name  originalGravity srm
0                    She WILL!            1.084  17
1    Defender American Pale Ale            1.044  22
2                        Hazel            1.036  19
3  Cinderellas Twin Double IPA           1.087  11
4              Independence Ale            1.048  14
```

```
In [3]: features = ["abv", "originalGravity"]

        X_train_dict = beer[features].to_dict(orient="records")
        y_train = beer["ibu"]
        val = beer_test
        X_val_dict = val[features].to_dict(orient="records")
        val["ibu"] = val["ibu"].fillna(0)
        y_val = val["ibu"]

        vec = DictVectorizer(sparse=False)
        vec.fit(X_train_dict)
        X_train = vec.transform(X_train_dict)
        X_val = vec.transform(X_val_dict)

        scaler = QuantileTransformer()
        scaler.fit(X_train)
        X_train_sc = scaler.transform(X_train)
        X_val_sc = scaler.transform(X_val)

        model = KNeighborsRegressor(n_neighbors=16)
        model.fit(X_train_sc, y_train)

        y_train_pred = model.predict(X_train_sc)
        y_val_pred = model.predict(X_val_sc)
        y_val_pred

Out[3]: array([ 45.71875,  32.1    ,  12.21875, ...,  40.5625 ,  25.0625 ,
                67.03125])

In [4]: df = pd.DataFrame(y_val_pred)
        df["id"] = beer_test.id
        df["ibu"] = df[0]
        df = df.drop(0, axis=1)
        df.to_csv("test.csv", header=True)
```

This was my first submission to kaggle. The strategy here was to get a feel for what the competition is like, so I used my model from Lab 5A.

```
In [5]: from sklearn.feature_extraction.text import TfidfVectorizer

        beer.description = beer.description.fillna("")

        vec = TfidfVectorizer(norm=None)
        vec.fit(beer["description"])
        tf_idf_sparse = vec.transform(beer["description"])
        tf_idf_sparse

        beer["new_description"] = pd.DataFrame(tf_idf_sparse.sum(axis=1))
```

```
beer_test.description = beer.description.fillna("")

vec = TfidfVectorizer(norm=None)
vec.fit(beer_test["description"])
tf_idf_sparse = vec.transform(beer_test["description"])
tf_idf_sparse

beer_test["new_description"] = pd.DataFrame(tf_idf_sparse.sum(axis=1))
```

```
In [6]: features = ["abv", "originalGravity", "new_description"]

        X_train_dict = beer[features].to_dict(orient="records")
        y_train = beer["ibu"]
        val = beer_test
        X_val_dict = val[features].to_dict(orient="records")
        val["ibu"] = val["ibu"].fillna(0)
        y_val = val["ibu"]

        vec = DictVectorizer(sparse=False)
        vec.fit(X_train_dict)
        X_train = vec.transform(X_train_dict)
        X_val = vec.transform(X_val_dict)

        scaler = QuantileTransformer()
        scaler.fit(X_train)
        X_train_sc = scaler.transform(X_train)
        X_val_sc = scaler.transform(X_val)

        model = KNeighborsRegressor(n_neighbors=16)
        model.fit(X_train_sc, y_train)

        y_train_pred = model.predict(X_train_sc)
        y_val_pred = model.predict(X_val_sc)
        y_val_pred

Out[6]: array([ 61.4375 ,  24.53125,  25.3375 , ...,  39.375  ,  19.075  ,  48.1    ])

In [7]: df = pd.DataFrame(y_val_pred)
        df["id"] = beer_test.id
        df["ibu"] = df[0]
        df = df.drop(0, axis=1)
        df.head()
        df.to_csv("test2.csv", header=True)
```

Here, I decided to start incorporating the description from the training data. I thought it would be best to use tf_idf and then sum across the columns, that way the higher the score, theoretically, the more unique words the description has. Then, my strategy was to use these description scores in the nearest neighbors model. As I used these features, I tweeked which scaler I used and how many neighbors to use.

```
In [8]: features = ["abv", "srm", "originalGravity", "new_description"]

        X_train_dict = beer[features].to_dict(orient="records")
        y_train = beer["ibu"]
        val = beer_test
        X_val_dict = val[features].to_dict(orient="records")
        val["ibu"] = val["ibu"].fillna(0)
        y_val = val["ibu"]

        vec = DictVectorizer(sparse=False)
        vec.fit(X_train_dict)
        X_train = vec.transform(X_train_dict)
        X_val = vec.transform(X_val_dict)

        scaler = QuantileTransformer()
        scaler.fit(X_train)
        X_train_sc = scaler.transform(X_train)
        X_val_sc = scaler.transform(X_val)

        model = KNeighborsRegressor(n_neighbors=7)
        model.fit(X_train_sc, y_train)

        y_train_pred = model.predict(X_train_sc)
        y_val_pred = model.predict(X_val_sc)
        y_val_pred
```

```
Out[8]: array([ 68.71428571,  34.14285714,  26.71428571, ...,  34.57142857,
                20.57142857,  63.71428571])
```

```
In [9]: df = pd.DataFrame(y_val_pred)
        df["id"] = beer_test.id
        df["ibu"] = df[0]
        df = df.drop(0, axis=1)
        df.to_csv("test3.csv", header=True)
```

After my second submission to kaggle, I thought it would be better to incorporate more quantitative variables along with the new description (represented by a number) and I consistently kept receiving rmse's around 20. When I submitted to kaggle my score was actually worse, so I began to think my model might be overfitting the data. So, I then decided to revert to something more basic and look at just one variable.

```
In [10]: features = ["originalGravity"]

         X_train_dict = beer[features].to_dict(orient="records")
         y_train = beer["ibu"]
         val = beer_test
```

```
X_val_dict = val[features].to_dict(orient="records")
val["ibu"] = val["ibu"].fillna(0)
y_val = val["ibu"]

vec = DictVectorizer(sparse=False)
vec.fit(X_train_dict)
X_train = vec.transform(X_train_dict)
X_val = vec.transform(X_val_dict)

scaler = QuantileTransformer()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

model = KNeighborsRegressor(n_neighbors=49)
model.fit(X_train_sc, y_train)

y_train_pred = model.predict(X_train_sc)
y_val_pred = model.predict(X_val_sc)
y_val_pred
```

```
Out[10]: array([ 45.13469388,  32.74693878,  20.16938776, ...,  26.92040816,
                 27.16938776,  61.29183673])
```

```
In [11]: df = pd.DataFrame(y_val_pred)
         df["id"] = beer_test.id
         df["ibu"] = df[0]
         df = df.drop(0, axis=1)
         df.to_csv("test4.csv", header=True)
```

With this model, I thought if I kept it more basic I could get a better feel for what the test data's ibus actually are. Like before, once I set the feature(s), I continued to manipulate the scaler and the number of neighbors to reach what I thought would be optimal It turned out that using just Original Gravity was better than my 2nd and 3rd model. At this point, I am thinking that the description will be the most useful variable in this data set to predict ibu's and that maybe it should be paired with Original Gravity to make a better model.

```
In [12]: features = ["originalGravity", "new_description"]

         X_train_dict = beer[features].to_dict(orient="records")
         y_train = beer["ibu"]
         val = beer_test
         X_val_dict = val[features].to_dict(orient="records")
         val["ibu"] = val["ibu"].fillna(0)
         y_val = val["ibu"]

         vec = DictVectorizer(sparse=False)
         vec.fit(X_train_dict)
         X_train = vec.transform(X_train_dict)
```

```
X_val = vec.transform(X_val_dict)

scaler = QuantileTransformer()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

model = KNeighborsRegressor(n_neighbors=1)
model.fit(X_train_sc, y_train)

y_train_pred = model.predict(X_train_sc)
y_val_pred = model.predict(X_val_sc)
y_val_pred
```

Out[12]: array([60., 45., 19., ..., 21., 16., 36.])

```
In [13]: df = pd.DataFrame(y_val_pred)
         df["id"] = beer_test.id
         df["ibu"] = df[0]
         df = df.drop(0, axis=1)
         df.to_csv("test5.csv", header=True)
```

My idea worked, I obtained a model with a low rmse using the 2 features I thought would be best. Turns out that my rmse was lowest only using 1 neighbor and continuing to use the Quantile Transformer. After submitting to kaggle, this submission turned out to be the worst submission. Honestly, at this point I have no clue what combination of variables will make my model best.

```
In [14]: features = ["originalGravity", "abv"]

         X_train_dict = beer[features].to_dict(orient="records")
         y_train = beer["ibu"]
         val = beer_test
         X_val_dict = val[features].to_dict(orient="records")
         val["ibu"] = val["ibu"].fillna(0)
         y_val = val["ibu"]

         vec = DictVectorizer(sparse=False)
         vec.fit(X_train_dict)
         X_train = vec.transform(X_train_dict)
         X_val = vec.transform(X_val_dict)

         scaler = QuantileTransformer()
         scaler.fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_val_sc = scaler.transform(X_val)

         model = KNeighborsRegressor(n_neighbors=24)
         model.fit(X_train_sc, y_train)
```

```
y_train_pred = model.predict(X_train_sc)
y_val_pred = model.predict(X_val_sc)
y_val_pred
```

```
Out[14]: array([ 49.52083333,  28.49166667,  16.89583333, ...,  36.5       ,
                 23.6125    ,  62.72916667])
```

```
In [15]: df = pd.DataFrame(y_val_pred)
         df["id"] = beer_test.id
         df["ibu"] = df[0]
         df = df.drop(0, axis=1)
         df.to_csv("test6.csv", header=True)
```

After the disappointment from submission 4, I thought it would be best to build on just Original Gravity. So, I added abv and continued to manipulate the number of neighbors. After submitting to kaggle, I finally beat my original submission, but at the time, it was not enough to move me up very far in the leaderboard. At this point, the competition has been going on for a while and there have been a lot more submissions that I noticed were below 20. Now, I am thinking that simply just using the given variables and using just KNeighborsRegressor with just parameter n will not be enough to make a better model. I started exploring the sci-kit learn documentation and learned of a new way to tweak my model that is shown in the next submission.

```
In [16]: features = ["abv", "srm", "originalGravity", "new_description"]

         X_train_dict = beer[features].to_dict(orient="records")
         y_train = beer["ibu"]
         val = beer_test
         X_val_dict = val[features].to_dict(orient="records")
         val["ibu"] = val["ibu"].fillna(0)
         y_val = val["ibu"]

         vec = DictVectorizer(sparse=False)
         vec.fit(X_train_dict)
         X_train = vec.transform(X_train_dict)
         X_val = vec.transform(X_val_dict)

         scaler = QuantileTransformer()
         scaler.fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_val_sc = scaler.transform(X_val)

         model = KNeighborsRegressor(n_neighbors=30, weights="distance")
         model.fit(X_train_sc, y_train)

         y_train_pred = model.predict(X_train_sc)
         y_val_pred = model.predict(X_val_sc)
         y_val_pred
```

```
Out[16]: array([ 57.85648864,  30.74634784,  26.48772882, ...,  36.25926126,
                 26.85897915,  58.32898557])
```

In [17]:
```
df = pd.DataFrame(y_val_pred)
df["id"] = beer_test.id
df["ibu"] = df[0]
df = df.drop(0, axis=1)
df.to_csv("test7.csv", header=True)
```

I found a new argument I could use in the KNeighborsRegressor that would weight the neighbors differently. Using the weights argument from my understanding puts more emphasis on the neighbors closer to each data point and makes a different prediction than the default weights. I got this information from sci-kit learn's documentation. After using weights="distance" on just originalGravity and abv I received a much lower rmse. I continued to add more variables and continued to change the number of neighbors until I reached this rmse. This must have over fit the training data because my kaggle submission didn't beat my previous score from submission 6.

In [18]:
```
features = ["originalGravity", "srm"]
X_train_dict = beer[features].to_dict(orient="records")
y_train = beer["ibu"]

vec = DictVectorizer(sparse=False)
scaler = QuantileTransformer()
model = KNeighborsRegressor(n_neighbors=30, metric="euclidean")
pipeline = Pipeline([("vectorizer", vec), ("scaler", scaler),
                     ("fit", model)])

vec.fit(X_train_dict)
X_train = vec.transform(X_train_dict)
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)

scores = cross_val_score(pipeline, X_train_dict, y_train, cv=10,
                         scoring="neg_mean_squared_error")
np.sqrt(np.mean(-scores))
```

Out[18]: 23.338396670922862

In [19]:
```
features = ["abv", "originalGravity", "new_description"]

X_train_dict = beer[features].to_dict(orient="records")
y_train = beer["ibu"]
val = beer_test
X_val_dict = val[features].to_dict(orient="records")
val["ibu"] = val["ibu"].fillna(0)
y_val = val["ibu"]

vec = DictVectorizer(sparse=False)
vec.fit(X_train_dict)
X_train = vec.transform(X_train_dict)
X_val = vec.transform(X_val_dict)
```

```python
scaler = QuantileTransformer()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

model = KNeighborsRegressor(n_neighbors=30, weights="distance",
                            metric="manhattan")
model.fit(X_train_sc, y_train)

y_train_pred = model.predict(X_train_sc)
y_val_pred = model.predict(X_val_sc)
y_val_pred
scores = cross_val_score(pipeline, X_train_dict, y_train, cv=10,
                         scoring="neg_mean_squared_error")
np.sqrt(np.mean(-scores))
```

Out[19]: 23.840531629305914

In [20]:
```python
df = pd.DataFrame(y_val_pred)
df["id"] = beer_test.id
df["ibu"] = df[0]
df = df.drop(0, axis=1)
df.to_csv("test8.csv", header=True)
```

In [21]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
```

In [22]:
```python
features = ["originalGravity", "abv"]

X_train_dict = beer[features].to_dict(orient="records")
y_train = beer["ibu"]
val = beer_test
X_val_dict = val[features].to_dict(orient="records")
val["ibu"] = val["ibu"].fillna(0)
y_val = val["ibu"]

vec = DictVectorizer(sparse=False)
vec.fit(X_train_dict)
X_train = vec.transform(X_train_dict)
X_val = vec.transform(X_val_dict)

scaler = QuantileTransformer()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

regr = RandomForestRegressor(max_depth=None, random_state=0,
                             n_estimators=100)
```

```
regr.fit(X_train_sc, y_train)
y_train_pred = regr.predict(X_train_sc)

scores = cross_val_score(pipeline, X_train_dict, y_train, cv=10,
                         scoring="neg_mean_squared_error")
print(np.sqrt(np.mean(-scores)))

y_val_pred = regr.predict(X_val_sc)
y_val_pred
```

```
22.4662026995
```

```
Out[22]: array([ 29.58871429,  37.91233467,  12.62970097, ...,  38.14276221,
                 27.85552889,  59.98551552])
```

```
In [23]: df = pd.DataFrame(y_val_pred)
         df["id"] = beer_test.id
         df["ibu"] = df[0]
         df = df.drop(0, axis=1)
         df.to_csv("test9.csv", header=True)
```

```
In [24]: from collections import Counter
```

```
In [25]: bag_of_words = (
             beer.loc[:100, "description"].
             str.lower().
             str.replace("[^A-Za-z\s]", "").
             str.split()
         ).apply(Counter)
```

```
In [26]: ipa = beer.description.str.contains("ipa")
         red = beer.description.str.contains("red")
         wheat = beer.description.str.contains("wheat")
         ale = beer.description.str.contains("ale")
         premium = beer.description.str.contains("premium")
         lager = beer.description.str.contains("lager")
         dark = beer.description.str.contains("dark")
         light = beer.description.str.contains("light")
         smooth = beer.description.str.contains("smooth")
         crisp = beer.description.str.contains("crisp")
         stout = beer.description.str.contains("stout")
         golden = beer.description.str.contains("golden")


         beer["ipa"] = ipa
         beer["red"] = red
         beer["wheat"] = wheat
```

```
beer["ale"] = ale
beer["premium"] = premium
beer["lager"] = lager
beer["dark"] = dark
beer["light"] = light
beer["smooth"] = smooth
beer["crisp"] = crisp
beer["stout"] = stout
beer["golden"] = golden

ipa = beer_test.description.str.contains("ipa")
red = beer_test.description.str.contains("red")
wheat = beer_test.description.str.contains("wheat")
ale = beer_test.description.str.contains("ale")
premium = beer_test.description.str.contains("premium")
premium = beer_test.description.str.contains("premium")
lager = beer_test.description.str.contains("lager")
dark = beer_test.description.str.contains("dark")
light = beer_test.description.str.contains("light")
smooth = beer_test.description.str.contains("smooth")
crisp = beer_test.description.str.contains("crisp")
stout = beer_test.description.str.contains("stout")
golden = beer_test.description.str.contains("golden")

beer_test["ipa"] = ipa
beer_test["red"] = red
beer_test["wheat"] = wheat
beer_test["ale"] = ale
beer_test["premium"] = premium
beer_test["lager"] = lager
beer_test["dark"] = dark
beer_test["light"] = light
beer_test["smooth"] = smooth
beer_test["crisp"] = crisp
beer_test["stout"] = stout
beer_test["golden"] = golden
```

In [27]:
```
features = ["originalGravity", "ipa", "red", "wheat",
            "abv", "isOrganic", "glass",
            "ale", "premium", "lager", "dark", "light",
            "smooth", "crisp",
            "stout", "golden"]

X_train_dict = beer[features].to_dict(orient="records")
y_train = beer["ibu"]
val = beer_test
X_val_dict = val[features].to_dict(orient="records")
val["ibu"] = val["ibu"].fillna(0)
```

```
y_val = val["ibu"]

vec = DictVectorizer(sparse=False)
vec.fit(X_train_dict)
X_train = vec.transform(X_train_dict)
X_val = vec.transform(X_val_dict)

scaler = QuantileTransformer()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

regr = RandomForestRegressor(max_depth=None, random_state=0, n_estimators=100)

regr.fit(X_train_sc, y_train)
y_train_pred = regr.predict(X_train_sc)

scores = cross_val_score(regr, X_train, y_train, cv=5,
                         scoring="neg_mean_squared_error")
print(np.sqrt(np.mean(-scores)))

y_val_pred = regr.predict(X_val_sc)
y_val_pred
```

```
23.7544473442
```

```
Out[27]: array([ 32.8575    ,  33.96369048,  14.03      , ...,  37.03216667,
                 28.96453333,  39.5177381 ])
```

```
In [28]: df = pd.DataFrame(y_val_pred)
         df["id"] = beer_test.id
         df["ibu"] = df[0]
         df = df.drop(0, axis=1)
         df.to_csv("test10.csv", header=True)
```

3 Submission Instructions

Once you are finished, follow these steps:

1. Restart the kernel and re-run this notebook from beginning to end by going to `Kernel > Restart Kernel and Run All Cells`.
2. If this process stops halfway through, that means there was an error. Correct the error and repeat Step 1 until the notebook runs from beginning to end.
3. Double check that there is a number next to each code cell and that these numbers are in order.

Then, submit your lab as follows:

1. Go to `File > Export Notebook As > PDF`.
2. Double check that the entire notebook, from beginning to end, is in this PDF file. (If the notebook is cut off, try first exporting the notebook to HTML and printing to PDF.)
3. Upload the PDF to PolyLearn.