Statistical Methods in Artificial Intelligence
Team 38: Restuarant Revenue Prediction

A Report By
Anjali Shenoy 201401114)
Rehas Sachdeva (201401102)
Saumya Rawat (201401110)

INTRODUCTION:

Many enterprises today are becoming highly motivated in the art of finding **anomalies, patterns and correlations** within the huge data sets generated by them, to predict outcomes. They want to improve their online reviews to attract clientele or seek to establish a new business that is **mindful of what drives good reviews.** This is particularly true for restaurants and food establishments.

Several studies have been conducted which look at the **correlation between a restaurant's success and its reviews and ratings** on Websites like Yelp, or data provided by companies like Tab Food Investments (TFI). TFI is behind the famous brands like Burger King, Sbarro, Popeyes etc, interested in extrapolating their data across geographies and cultures.

We will be working with a **TFI data set of about 1 lakh Turkish restaurants.**

PROBLEM STATEMENT:

"To develop a model and a set of preprocessing procedures to accurately predict the annual restaurant sales of 100,000 regional locations using various parameters."

We classify this problem as a **supervised learning problem.**

DATASET DESCRIPTION:

- The dataset represents ~100,000 Turkish restaurants as uploaded on Kaggle. The size of the training dataset is 137 samples whereas the size of the test set is 100,000 samples.
- The 43 attributes of the dataset are:
  - **Id**: Restaurant ID.
  - **Open Date**: Date that the restaurant opened in the format M/D/Y
  - **City**: The city name that the restaurant resides in
  - **City Group**: The type of city can be either big cities or other
  - **Type**: The type of the restaurant where **FC** - Food Court, **IL** - Inline, **DT** - Drive through and **MB** - Mobile.
  - **P-Variables (P1, P2, ... ,P37):** Obfuscated variables within three categories[1]:
    - Demographic data, e.g population, age, gender
    - real estate data e.g car park availability and front facade
    - commercial data e.g points of interest, other vendors, etc.
  - **Revenue**: Annual revenue of a restaurant in a given year. This is the target to be predicted.

[1] It is unknown if each variable contains a combination of the three categories or are mutually exclusive.

FEATURE EXPLANATION:

- **Sales depend on the location and type of city** it is in. If the city is a metropolitan city, it will have a larger customer base than a town, and hence revenue generated will be different in both these cities.
- On a similar note, revenues generated will be different for **different restaurant types**- Drive through will attract more customers in a remote area where as a restaurant will attract more customers if it is placed in the heart of the city.
- The open date attribute doesn't do much as such. But if processed to get number of days a restaurant stays open, year of opening, month of opening, we can get an idea of **cyclical or seasonal patterns** that affect revenue.
- Apart from these, we have demographic attributes e.g population, age, gender; real estate data e.g car park availability and front facade; and commercial data e.g points of interest, other vendors, etc. These also decide sales to varied extents. They can also be **correlated.**

CHALLENGES:

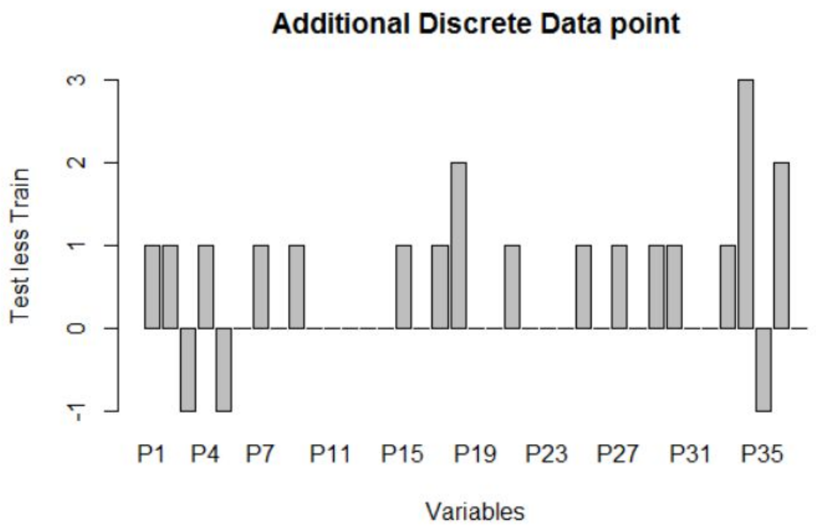
1. The size of training dataset is 137 samples while that of test dataset is 1,00,000 samples. This is a **large disparity**.
2. **We don't have the ground truth for our test data**. So we cannot use sophisticated performance measures like precision, recall, k-fold cross validation etc. We only know RMSE for the entire test data, via submission on Kaggle.

3. **Whether to predict revenue, log(revenue) or sqrt(revenue)** as we see that training data follows normal distribution when taken with log(revenue). But we can't say the same about test dataset.

4. **Parsing** the data types of various attributes.

5. **Unaccountability problem:** the **disparity between the features for the training set and test set** as the test set contains more information than the training set. Type attribute spans FC, IL and DT within the training set but is missing MB which is present within the test set. Likewise, there are 34 cities in the training set but 57 in the test set. Most supervised learning models fitted through the training set will encounter errors due to **missing coefficients for the additional classes.**

6. **Categorical vs continuous problem:** While it is clear that certain variables such as City and Type can be treated as categorical, it is unclear **whether the obfuscated P-Variables should be treated categorical** as they are discrete in nature. The unaccounted issue persists if they are treated as categorical. For example, if we plot the difference between the number of unique values in the test set and the training set for each P-Variables, there are almost always more values in the test set than the training set.



7. **Zero problem:** For certain P-Variables, a large number of samples contain zero values and **are dependent among each other such that if one p-variable has zero on a certain row, the probability that other p-variables take on a zero value is high.** For the train dataset, this probability is 1. These could be missing values or simply be highly correlated.
8. **Non-linear Regression Problem:** is possibly the **most challenging among**, 2 class classification, multi class classification, non-linear classification and non-linear regression **class of problems**.

LANGUAGES & TOOLKITS:

- Python (Jupyter Notebook)
- SKLearn toolkit

FEATURE EXTRACTION & SELECTION:

The training and test data are available as CSV files, containing a total of ~100,000 samples of Turkish Restaurants. The test data is too large compared to the train data and majority of the data fields are obfuscated variables without giving any prior knowledge of each one.

Our pre-processing is done as follows:

1. **Parsing Open Date:** Opening date cannot simply be assumed to be a factor, so two treatments were carried out. The first treatment is to calculate the **number of days open** by taking the difference, in days, between the opening date and an arbitrary constant such that it is later than the latest opening date of all samples. The date chosen is January 1, 2015 and the difference in days can be treated as a continuous feature. The second treatment is to create **two additional features where one is the month that they opened and the other is the year** that they opened. These two features can potentially help proxy seasonality differences since restaurant revenues are highly cyclical.

```
def parse_date(data):

    # Assume a date which is later than latest opening date of all the restaurants
    latest_date = datetime.strptime("01/01/2015", '%m/%d/%Y')

    open_num_days = []
    open_month = []
    open_year = []

    for date in data['Open Date']:
        cur_date = datetime.strptime(date, '%m/%d/%Y')
        open_num_days.append((latest_date - cur_date).days)
        open_month.append(cur_date.month)
        open_year.append(cur_date.year)
```

```
data['Days'] = open_num_days
data['Month'] = open_month
data['Year'] = open_year

del data["Open Date"]
return data
```

Above is the technique proposed in the paper. However, we didn't stop there. We found that **log transform** on these three new features improved accuracy. Furthermore, we found that not considering the attributes 'Month' and 'Year', **but only using 'days open' with log transform**, further improved accuracy. This is probably due to the **overfitting** to training data caused by using these 'Month' and 'Year' attributes.

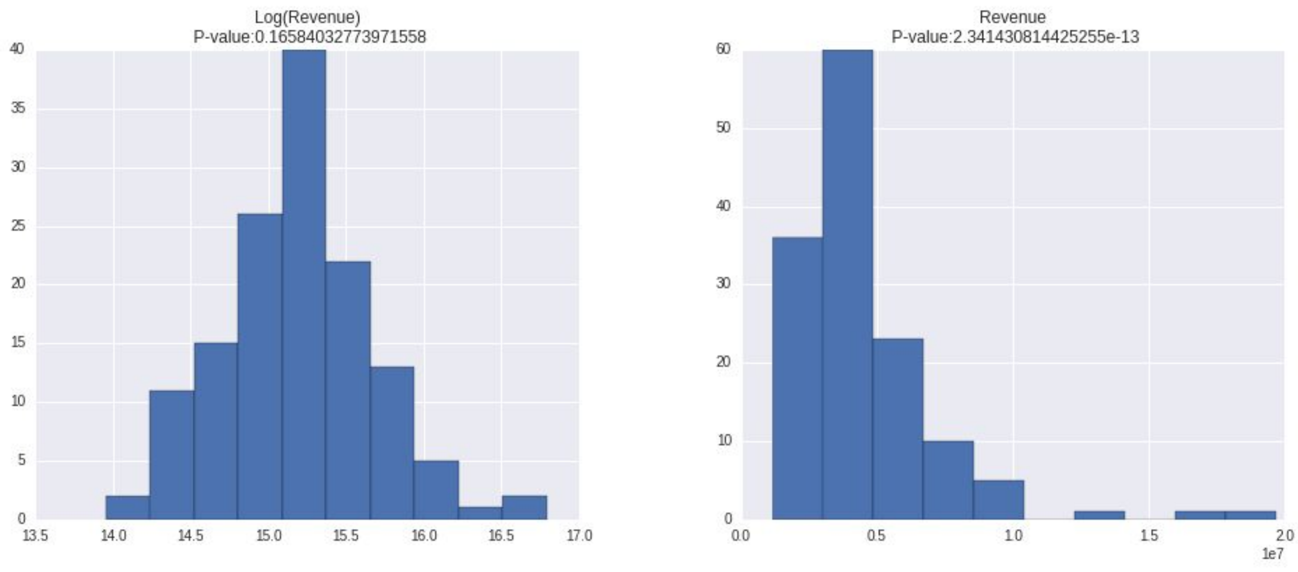
```
data['Days'] = data['Days'].apply(np.log)
data['Month'] = data['Month'].apply(np.log)
data['Year'] = data['Year'].apply(np.log)
```

```
del data["Month"]
del data["Year"]
```

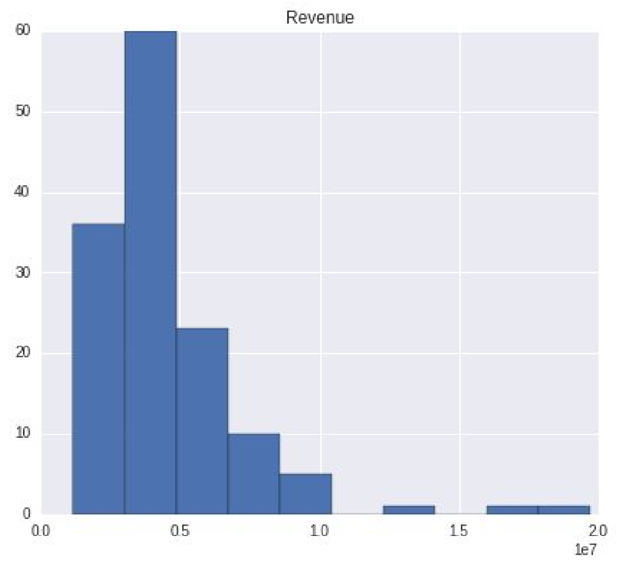
2. **Modelling an inherent distribution in Revenue variable:** We use **histograms** to see that log(revenue) follows an approximately **normal distribution**. We measure this using Shapiro's P value. So choosing target variable as **log(revenue)** instead of revenue improves performance of base models. Interested by this idea mentioned in the paper, we tried other **tricks like Sqrt treatment** on Revenue. The basic idea is that such transforms **reduce the skewness in the distributions of data.** We were surprised to learn

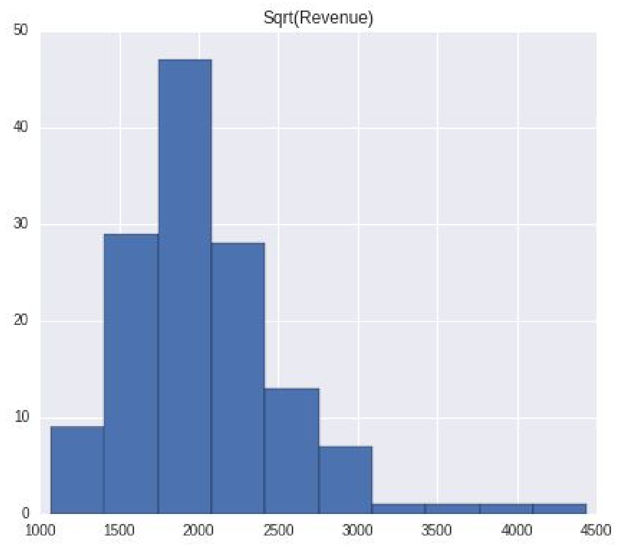
that Sqrt treatment gave much better results. **The maximum and minimum predictions made on test data were more extreme in case of Sqrt treatment**, so it probably gave better results for extreme points in test data.

```
# According to the paper
# Check distribution of revenue and log(revenue)
plt.rcParams['figure.figsize'] = (16.0, 6.0)
pvalue_before = shapiro(train["Revenue"])[1]
pvalue_after = shapiro(np.log(train["Revenue"]))[1]
graph_data = pd.DataFrame(
        {
            ("Revenue\n P-value:" + str(pvalue_before)) : train["Revenue"],
            ("Log(Revenue)\n P-value:" + str(pvalue_after)) : np.log(train["Revenue"])
        }
    )
graph_data.hist()

# log transform revenue as it is approximately normal. If this distribution for
# revenue holds in the test set, log transforming the variable before training
# models will improve performance vastly. However, we cannot be completely
# certain that this distribution will hold in the test set.
train["Revenue"] = np.log(train["Revenue"])
```



```
# Sqrt  treatment before after distributions
plt.rcParams['figure.figsize'] = (16.0, 6.0)
graph_data = pd.DataFrame(
      {
         ("Revenue") : train["Revenue"],
         ("Sqrt(Revenue)") : np.sqrt(train["Revenue"])
      }
   )
graph_data.hist()

# Proposed sqrt treatment for revenue
# Scale revenue by sqrt.
# The purpose is to decrease the influence of the few very large revenue values.
train["Revenue"] = train.Revenue.apply(np.sqrt)
```



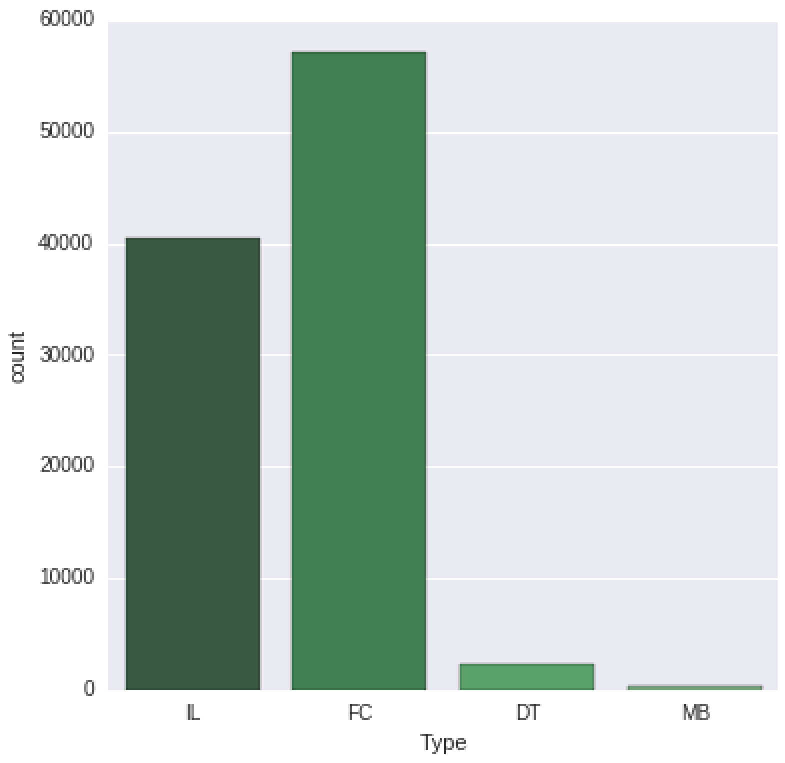


3. **Dealing with Type categorical variable:** Since the restaurant 'Type'- 'MB' is not present in the training but present in the Test, the Test set is modified so each mobile type restaurant is matched with a non-mobile type restaurant through finding the most similar features, as measured by euclidean distance **(KNN)**. We first construct a **query matrix** within the test set where each row is a mobile type restaurant. The **search matrix** consists of a test set where each row is not a mobile type restaurant. Any factor variables are removed from the query since the **KNN algorithm** within our implementation can only deal with continuous variables. We also tried other classifiers for this purpose, like **Extra Trees Classifier**. An "extra trees" classifier, otherwise known as an "Extremely randomized trees" classifier, is a variant of a random forest. Unlike a random forest, at each step the entire sample is used and decision boundaries are picked at random, rather than the best one.

```
# Proposed alternative for Type treatment
# Two of the four Restaurant Types (DT and MB), are extremely rare
sns.countplot(x='Type', data=data, palette="Greens_d")
```

```
# One hot encode Restaurant Type
data = data.join(pd.get_dummies(data['Type'], prefix="T"))

# Drop the original column. Also drop the extremely rare restaurant types.
data = data.drop(["Type", "T_MB", "T_DT"], axis=1)

# Map values for the very rare restaurant types to one of the common types

# tofit are the rows in the train set that belong to one of the common restaurnat
types
tofit = data[data.index<num_train]
tofit = tofit.ix[((tofit.T_FC == 1) | (tofit.T_IL == 1))]

# tofill are rows in either train or test that belong to one of the rare types
tofill = data.ix[((data.T_FC == 0) & (data.T_IL == 0))]

# Restaurants with type FC are labeled 1, those with type IL are labeled 0.
y = tofit.T_FC

# Drop the label columns
X = tofit.drop(["T_FC", "T_IL"], axis=1)

# Define and train a model to impute restaurant type
# The grid below just has a range of values that commonly
# work well with random forest type models (of which ExtraTrees is one).
model_grid = {'max_depth': [None, 8], 'min_samples_split': [4,9,16],
              'min_samples_leaf':[1,4], 'max_features':['sqrt', 0.5, None]
             }
type_model = ExtraTreesClassifier(n_estimators=25, random_state=0)

grid = RandomizedSearchCV(type_model, model_grid, n_iter=10, cv=5,
scoring="roc_auc")
grid.fit(X, y)

type_model.set_params(**grid.best_params_)
type_model.fit(X, y)
```

```
imputations = type_model.predict(tofill.drop(["T_FC", "T_IL"], axis=1))
data.loc[(data.T_FC == 0) & (data.T_IL == 0), "T_FC"] = imputations
data = data.drop(["T_IL"], axis=1)
```



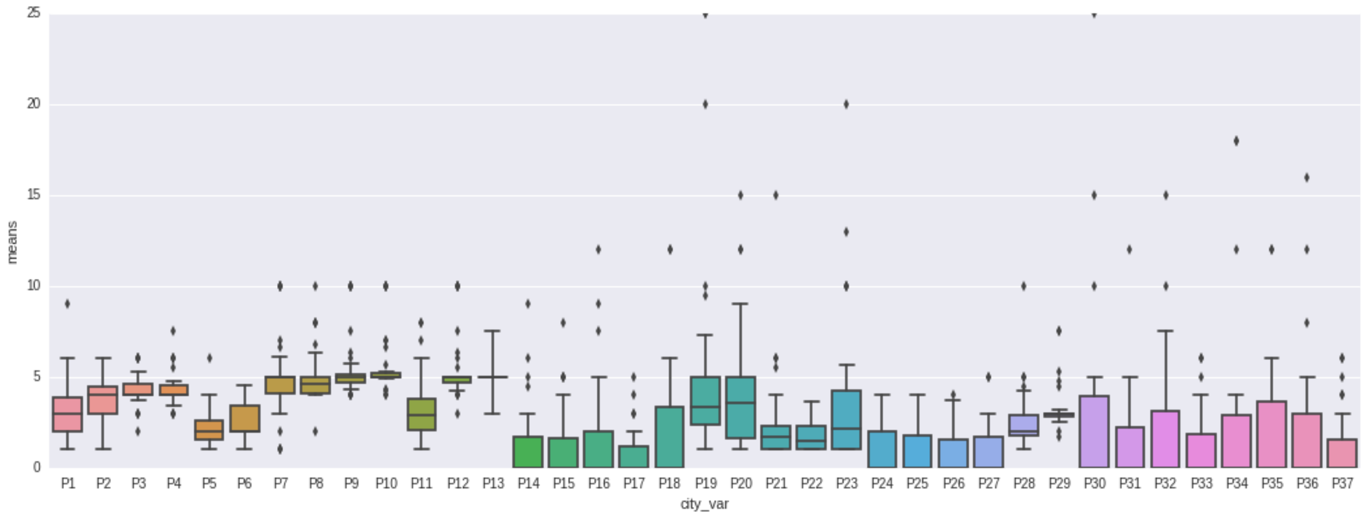
But **KNN** performed slightly better.

```
# According to the paper
def adjust_type(data):

    # Augment data with id for original order of records
    data.loc[:, "tempIdx"] = data.index

    # Get records with restaurant type = "Mobile"
    query_matrix = data.loc[data.Type == "MB", :]

    # Get records with restaurant type != "Mobile"
    search_matrix = data.loc[data.Type != "MB", :]

    # Use only continuous features for classification
    features = data.columns.values[2:]

    # Predict known type using KNN
    clf = KNeighborsClassifier(n_neighbors=3)
    clf.fit(search_matrix.loc[:, features], search_matrix.loc[:, 'Type'])
    query_matrix.loc[:, 'Type'] = clf.predict(query_matrix.loc[:, features])

    # Construct data to original form and order
    data = pd.concat((search_matrix.loc[:, "Type" : "tempIdx"],
                query_matrix.loc[:, "Type" : "tempIdx"]), ignore_index = True)
    data = data.sort_values(["tempIdx"])
    del data["tempIdx"]

    # One hot encode Type
    data = data.join(pd.get_dummies(data['Type'], prefix="T"))

    # Since only n-1 columns are needed to binarize n categories, drop one of the
new columns.
    # And drop the original columns.
    data = data.drop(["Type", "T_IL"], axis=1)

    return data

# Convert unknown restaurant types in test data to known restaurant types using
KNN
data = adjust_type(data)
```

4. **Dealing with City categorical variable:** This problem, much like type, can also be treated with the kNN approach. We instead introduce a **K-Means clustering methodology** in imputing City values which is similar in nature to the kNN treatment. The unsupervised algorithm looks for tight clusters of data in features to and assigns them to an arbitrary cluster. Since we know that

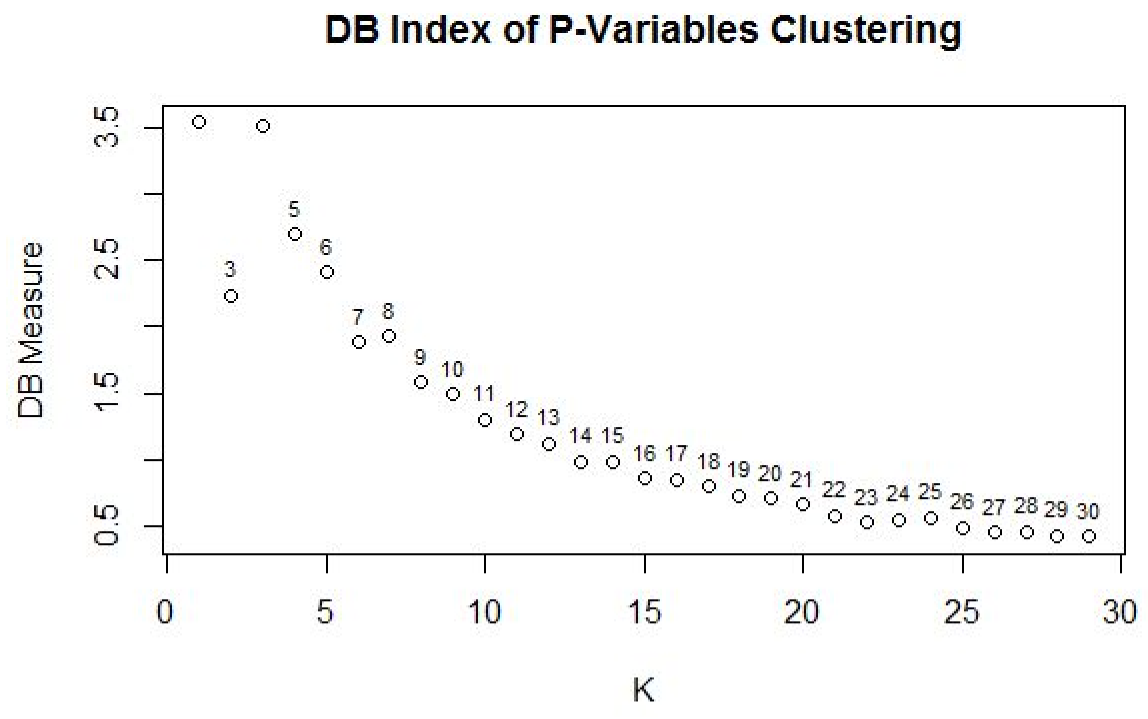
the P-Variables are supposed to represent three categories including geographical data, we assume each P-Variable to be mutually exclusive in category and attempt to identify variables that best represent City which should contain many geographical attributes. More specifically, we collect the **average P-Variable for each subset of cities and plot the deviation over all cities.** The intuition behind this idea is simple. Since we do not know exactly what each p-variable represents, under the assumption of mutually exclusive categories, a change in city should elicit a change in certain p-variables. Plotting the mean over each city should help us identify that through examining the range of deviation a variable can have. One caveat to this method is related to the continuous vs. categorical problem where p-variables are discrete in its values and large deviations are simply a result of its categorical values.

```
# According to the paper
# There is unaccounted problem for City as well.
# Plotting mean of P-variables over each city helps us see which P-variables are
highly related to City
# since we are given that one class of P-variables is geographical attributes.
distinct_cities = train.loc[:, "City"].unique()

# Get the mean of each p-variable for each city
means = []
for col in train.columns[4:41]:
    temp = []
    for city in distinct_cities:
        temp.append(train.loc[train.City == city, col].mean())
    means.append(temp)

# Construct dataframe for plotting
city_pvars = pd.DataFrame(columns=["city_var", "means"])
for i in range(37):
    for j in range(len(distinct_cities)):
        city_pvars.loc[i+37*j] = ["P"+str(i+1), means[i][j]]

# Plot boxplot
plt.rcParams['figure.figsize'] = (18.0, 6.0)
sns.boxplot(x="city_var", y="means", data=city_pvars)

# From this we observe that P1, P2, P11, P19, P20, P23, and P30 are approximately a
good
# proxy for geographical location.
```



From this we observe that P1, P2, P11, P19, P20, P23, and P30 are approximately a good proxy for geographical location. Using k-means methodology, we can reclassify each 137 restaurant cities into arbitrary clusters based on these variables.

To select an optimal K, we use the **Davies and Bouldin index** which is a measure of the ratio of **within-cluster distance and between-cluster distance.** Under optimality, the DB measure should be extremely low as the spread within-cluster is small and the distance between-clusters is large. It is clear that the DB Index converges and selecting K between the range of 20 to 25 would be optimal. In our use case, we set K = 20.

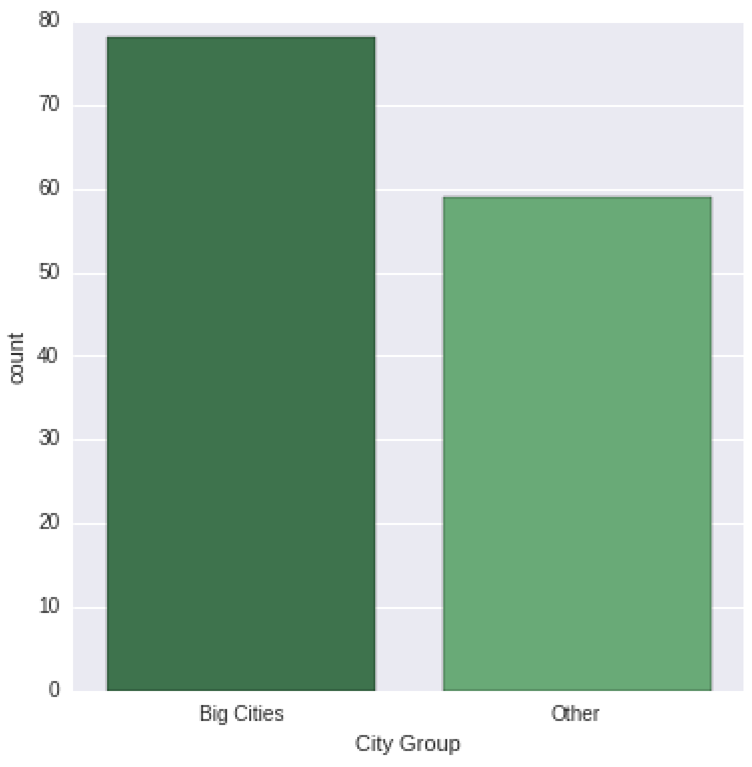


```
# K Means treatment for city (mentioned in the paper)
def adjust_cities(data, train, k):

    # As found by box plot of each city's mean over each p-var
    relevant_pvars =  ["P1", "P2", "P11", "P19", "P20", "P23", "P30"]
    train = train.loc[:, relevant_pvars]

    # Optimal k is 20 as found by DB-Index plot
    kmeans = cluster.KMeans(n_clusters=k)
    kmeans.fit(train)

    # Get the cluster centers and classify city of each data instance to one of the
centers
    data['City Cluster'] = kmeans.predict(data.loc[:, relevant_pvars])
    del data["City"]
    return data

# Convert unknown cities in test data to clusters based on known cities using
KMeans
data = adjust_cities(data, train, 20)
```

Above treatment was mentioned in the paper and did give significant improvement in results. However, we tried another approach which gave even better results. Notice that the unaccountability problem can also be solved if **we replace the city name by its count over the entire dataset.** The unaccounted cities will simply have lower counts and the classifier will learn accordingly. Our results show that this treatment gave **slightly better results.**

```
# Proposed treatment for City
# Replace city names with count of their frequency
city_counts = data["City"].value_counts()
data["City"] = data["City"].replace(city_counts)
```

5. **Dealing with City Group categorical variable:** In this case we simply use one hot encoding as the two categories of City Group both appear very frequently.



```
# According to the paper
# The two categories of City Group both appear very frequently
plt.rcParams['figure.figsize'] = (6.0, 6.0)
sns.countplot(x='City Group', data=train, palette="Greens_d")

# One hot encode City Group
data = data.join(pd.get_dummies(data['City Group'], prefix="CG"))

# Since only n-1 columns are needed to binarize n categories, drop one of
the new columns.
# And drop the original columns.
data = data.drop(["City Group", "CG_Other"], axis=1)
```

6. **Dealing with P-Variables:** We use **PCA** for the P-Variables because we aren't given exactly what these represent. They can be correlated. So PCA can represent them better. But the results did not improve with PCA treatment, as shown in the paper itself.

Further, the paper could not come up with any treatment for the Zero Problem that actually improved results. So we tried an alternative treatment which marginally improved results. This was adding an **additional attribute called 'zeros' to store the count of the columns which have the value 0, among the columns with this behaviour.**

```
# Proposed idea, in addition to the paper
# A certain set of columns are either mostly all zero or all non-zero.
# We added a feature to mark this  - The "zeros" feature that holds this count of
zero columns.

# The features with the notable zero behavior:
zero_cols = ['P14', 'P15', 'P16', 'P17', 'P18', 'P24', 'P25', 'P26', 'P27', 'P30', 'P31', 'P32', 'P33',
            'P34', 'P35', 'P36', 'P37']

data['zeros'] = (data[zero_cols] == 0).sum(1)
```

Moreover the paper considers the P-variables as continuous. But we found that taking them as categorical and processing them with one-hot encoding, improved the results by a large margin.

```
# Proposed alternative to categorical vs continuous problem
# one hot encoding for "P" variables - taking them as categorical
for col in data.columns:
    if col[0] == 'P':
        data = data.join(pd.get_dummies(data[col], prefix=col))
        # Since only n-1 columns are needed to binarize n categories,
        # drop one of the new columns.
        data = data.drop([col, data.columns[-1]], axis=1)
```

7. **Additional Pre-Processing:** We also tried **scaling all the numerical attributes between 0 and 1 (normalization).** This was not mentioned in the paper, but it is a very common technique, to

enable a classifier to perform better. As expected, this improved the results by a large margin.

```
# Proposed idea, in addition to the paper
# Scale all input features to between 0 and 1.
min_max_scaler = MinMaxScaler()

data = pd.DataFrame(data=min_max_scaler.fit_transform(data),
                    columns=data.columns, index=data.index)
```

MODELS:

1. **Random Forest:** One of the models we are focusing on is random forest. Random forests are promising because they have desirable characteristics such as **running efficiently on large datasets, and flexibility**, having been used effectively for a variety of applications, including **multi class image recognition, 3D face analysis, land classification and disease prediction etc.**
    - A random forest is an **ensemble of decision trees** created using **random variable selection and bootstrap aggregating** (bagging). What this means is that first a group of decision trees are created.
    - For each individual tree, **a random sample with replacement of the training data is used for training.** Also, at each node of the tree, the split is created by only looking at a random subset of the variables. A commonly used number for each split is the square root of the number of variables. The prediction is made by **averaging the predictions of all the individual trees.**
    - Random forests can also provide an error estimate, called the **out-of-bag error.** This is computed by feeding the individual trees cases that they haven't seen. Because each

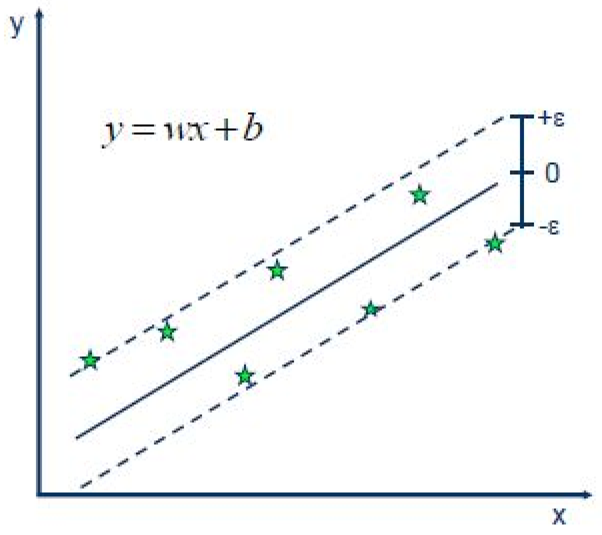
tree is created with a bootstrap sample, we expect about 1/3 of the samples to not be included in the training for any given tree. The training data that was not included in the bootstrap sample for a given tree is fed through that tree, and a prediction is made. The results of doing this for all trees are computed, and for each sample (row) an out of bag estimate is created by averaging the results of all the trees. The proportion of incorrect classification gives an estimate of the error rate.

- Another important aspect of random forests is their ability to **rank variable importance.** The variable importance is computed by finding the out-of-bag accuracy, and then by using a permutation test on the variable, by randomly permuting the out-of-bag values for a given variable, and then measuring the accuracy again. The difference in accuracy gives an estimate of the importance of the variable.

```
# RF model
rf = RandomForestRegressor()
rf.fit(train_processed, train["Revenue"])
results_rf = rf.predict(test_processed)
```

2. **SVM:** Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences.
    - In the case of regression, **a margin of tolerance (epsilon)** is set in approximation to the SVM which would have already requested from the problem.

- The main idea is always the same: **to minimize error, individualizing the hyperplane which maximizes the margin,** keeping in mind that part of the error is tolerated.

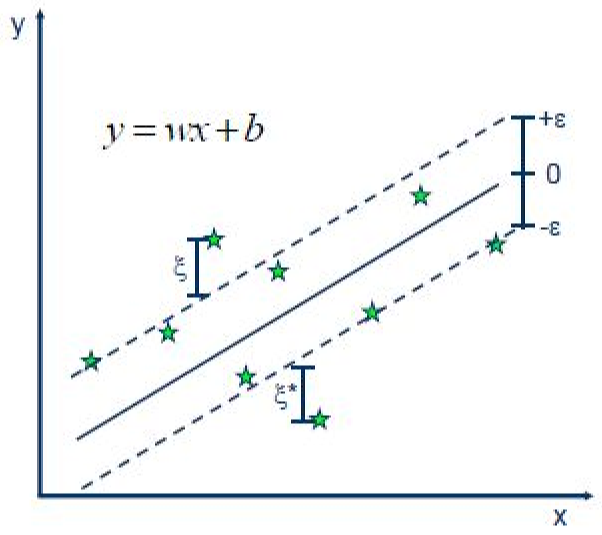


- Solution:

$$\min \frac{1}{2}\|w\|^2$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon$$
$$wx_i + b - y_i \le \varepsilon$$



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$
$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \ge 0$$

```
# SVR model
lin_clf = SVR()
lin_clf.fit(train_processed, train["Revenue"])
results_svm = lin_clf.predict(test_processed)
```

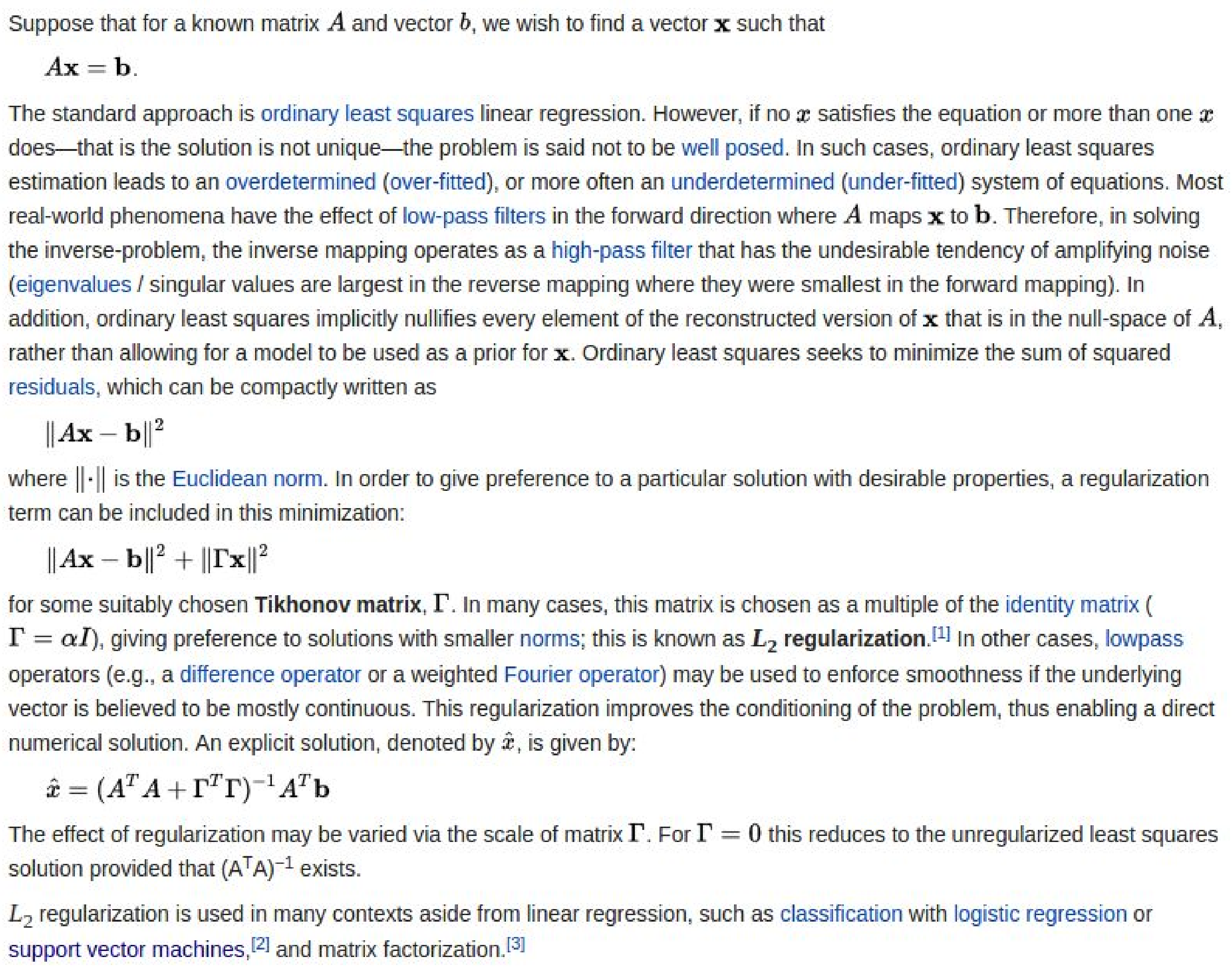
3. **Ensembling:** In machine learning problems a common way to improve accuracy is through **ensembling multiple prediction algorithms.** Ensemble learning involves creating base prediction algorithms, and then combining them in a way that leads to better accuracy. Ensemble models can perform better than the models they are composed of if the models are independent, and if they get greater than 50% accuracy. This intuitively makes sense, as it should not be possible to increase accuracy by combining identical models, or by adding models that are not useful.**When errors of classifiers are uncorrelated, majority voting using many classifiers reduces error rates.**

```
# Try out different ensembles
# Ensembling helps to average out the error due to different models
# Paper
results_agg = 0.5*results_rf + 0.5*results_svm
```

```
# Ensembling with Ridge (explained below)
# More weight to Ridge as it gave the best individual results, then to random
forest and then to SVR
results_agg = 0.3*results_rf + 0.2*results_svm + 0.5*results_ridge
```

4. **Ridge Model.** This model is not mentioned in the paper but nevertheless we wanted to try it, intrigued by the ensembling method. It was totally worth it as we ended up getting a huge improvement in the results. Even though **Ridge as a standalone model gave better results compared to Random Forest and SVR but an ensemble of the three gave the best results among all our attempts.** Ensembling helps to **average out the error** due to different models.

   Tikhonov regularization is known as ridge regression. The following derivation is from Wikipedia.



Suppose that for a known matrix A and vector b, we wish to find a vector $\mathbf{x}$ such that

$$A\mathbf{x} = \mathbf{b}.$$

The standard approach is ordinary least squares linear regression. However, if no x satisfies the equation or more than one x does—that is the solution is not unique—the problem is said not to be well posed. In such cases, ordinary least squares estimation leads to an overdetermined (over-fitted), or more often an underdetermined (under-fitted) system of equations. Most real-world phenomena have the effect of low-pass filters in the forward direction where A maps $\mathbf{x}$ to $\mathbf{b}$. Therefore, in solving the inverse-problem, the inverse mapping operates as a high-pass filter that has the undesirable tendency of amplifying noise (eigenvalues / singular values are largest in the reverse mapping where they were smallest in the forward mapping). In addition, ordinary least squares implicitly nullifies every element of the reconstructed version of $\mathbf{x}$ that is in the null-space of A, rather than allowing for a model to be used as a prior for $\mathbf{x}$. Ordinary least squares seeks to minimize the sum of squared residuals, which can be compactly written as

$$\|A\mathbf{x} - \mathbf{b}\|^2$$

where $\|\cdot\|$ is the Euclidean norm. In order to give preference to a particular solution with desirable properties, a regularization term can be included in this minimization:

$$\|A\mathbf{x} - \mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2$$

for some suitably chosen **Tikhonov matrix**, Γ. In many cases, this matrix is chosen as a multiple of the identity matrix ($\Gamma = \alpha I$), giving preference to solutions with smaller norms; this is known as **L_2 regularization.**[1] In other cases, lowpass operators (e.g., a difference operator or a weighted Fourier operator) may be used to enforce smoothness if the underlying vector is believed to be mostly continuous. This regularization improves the conditioning of the problem, thus enabling a direct numerical solution. An explicit solution, denoted by $\hat{x}$, is given by:

$$\hat{x} = (A^TA + \Gamma^T\Gamma)^{-1}A^T\mathbf{b}$$

The effect of regularization may be varied via the scale of matrix Γ. For $\Gamma = 0$ this reduces to the unregularized least squares solution provided that $(A^TA)^{-1}$ exists.

L_2 regularization is used in many contexts aside from linear regression, such as classification with logistic regression or support vector machines,[2] and matrix factorization.[3]

```
# Proposed idea
# Ridge model
model_grid = [{'normalize': [True, False], 'alpha': np.logspace(0,10)}]
ridge_clf = Ridge()

# Use a grid search and leave-one-out CV on the train set to find the best
regularization parameter to use.
grid = GridSearchCV(ridge_clf, model_grid, cv=LeaveOneOut(train.shape[0]),
scoring='mean_squared_error')
grid.fit(train_processed, train["Revenue"])

# Re-train on full training set using the best parameters found in the last step.
ridge_clf.set_params(**grid.best_params_)
ridge_clf.fit(train_processed, train["Revenue"])

results_ridge = ridge_clf.predict(test_processed)
```

5. **Lasso Model:** The LASSO (**Least Absolute Shrinkage and Selection Operator**) is a regression method that involves penalizing the absolute size of the regression coefficients. By penalizing (or equivalently **constraining the sum of the absolute values of the estimates**) you end up in a situation where some of the parameter estimates may be exactly zero. The **larger the penalty applied, the further estimates are shrunk towards zero.** This is convenient when we want some automatic feature/variable selection, or when dealing with highly correlated predictors, where standard regression will usually have regression coefficients that are 'too large'.

   Give a set of input measurements x_1, x_2 ...x_p and an outcome measurement y, the lasso fits a linear model:

$$\hat{y}=b_0+b_1*x_1+b_2*x_2+\ldots b_p*x_p$$

   The criterion it uses is: **Minimize**:

$$\sum(y-\hat{y})^2 \text{ subject to } \sum|b_j|\leq s$$

   The first sum is taken over observations (cases) in the dataset. The bound "s" is a tuning parameter. When "s" is large enough, the constraint has no effect and the solution is just the usual multiple linear least squares regression of y on x1, x2, ...xp. However, for smaller values of s (s>=0) the solutions are shrunken versions of the least squares estimates. Often, some of the coefficients bj are zero. Choosing "s" is like choosing the number of predictors to use in a regression model, and

cross-validation is a good tool for estimating the best value for "s".

```
# Proposed idea
# Lasso model
model_grid = [{'normalize': [True, False], 'alpha': np.logspace(0,10)}]
lasso_clf = Lasso()

# Use a grid search and leave-one-out CV on the train set to find the best
regularization parameter to use.
grid = GridSearchCV(lasso_clf, model_grid, cv=LeaveOneOut(train.shape[0]),
scoring='mean_squared_error')
grid.fit(train_processed, train["Revenue"])

# Re-train on full training set using the best parameters found in the last step.
lasso_clf.set_params(**grid.best_params_)
lasso_clf.fit(train_processed, train["Revenue"])

results_lasso = lasso_clf.predict(test_processed)
```

VALIDATION TECHNIQUES USED:

We use **histograms** to see that log(revenue) follows an approximately normal distribution so choosing target variables as log(revenue) instead of revenue improves performance of base models.

We also visualize a **box plot of mean of P-variables** for each city to help us identify which P-variables are majorly geographical.

We also apply **Davies-Bouldin index for K-Means** clustering on variables: P1, P2, P11, P19, P20, P23, P30 to help us validate the best K to be used as number of clusters.

PERFORMANCE METRICS:

Since we **do not have the ground truth for the validation set**, we mainly tested out performance **by submitting our code on kaggle** to see what rank we got. This also gave us the **RMSE** values for the dataset and hence we used this as a **parameter for minimisation.**

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Where $\hat{y}_i$ is the predicted revenue of the ith restaurant and y_i is the actual revenue of the i^{th} restaurant.

ANALYSIS FROM THE PAPER:

- **Pre-processing on Open Date, City and Type** greatly improves performance over baseline models.
- **Log transformation on revenue** does not improve real test set performance although training set results are very promising.
- **Treating zero problem with PCA or KNN** doesn't improve performance probably due to large misspecification errors of the treatment models that introduce more noise rather than clarity.
- **Ensemble model of SVR and Random Forest** results in best overall results.
- Apart from zero problem treatment, **all solutions proposed in the paper** together lead to least RMSE.

ENHANCING THE IDEAS IN THE PAPER:

We wanted to explore further than the methods proposed in the paper, and hence we looked up different models of regression. We also tried different kinds of preprocessing on the data types:

- **Date Parsing:** Taking hint from the **log transform** on revenue to reduce skewness of data, we tried the same treatment for other numerical attributed like Days open, month, year etc. Doing this for Days open improve the results. We also observed that contrary to the paper's claim that Month of opening and year of opening can possibly help learn the effect of cyclical or seasonal patterns in revenue. We found that dropping these attributes all together, improved the accuracy.
- For restaurant type such as T_MB, T_DT which were very rare we dropped it from the table and substituted those values with predicted type using **Extra Trees classifier** instead of KNN treatment proposed in the paper. This gave us improved results in train dataset but the **submission RMSE for slightly higher than that of KNN treatment.**
- For the categorical and continuous problem we did a **one hot encoding** for "P" variables and took them as categorical. This greatly improved accuracy.
- We also **scaled all input features** between 0 and 1. This greatly improved accuracy. This is particularly important for Ridge and SVR models.
- We tried different ways of transforming the revenue to reduce skewness, apart from log(revenue), like **sqrt of revenue**, etc. This gave a mark improvement.
- **Zero problem:** A certain set of columns are either mostly all zero or all non-zero. We **added a feature** to mark this, storing the count of zero columns. This also greatly improved accuracy.
- We tried an alternative treatment for City problem, **replacing cities with their total counts**. This also greatly improved accuracy.
- We also looked at the **ridge model** and tested it as an ensemble with SVM, Lasso and random forest, and as a standalone classifier. Ridge as a standalone **gave us even better results than**

the methods mentioned in the paper and a Kaggle rank of 7, with corresponding **RMSE score of 1,749,468.**

- We also looked at the **lasso model** and tested it as an ensemble with SVM, Lasso and random forest, and as a standalone classifier. Lasso as a standalone performed better than Random Forest and SVR as standalones, but **not as good as the ensemble approaches with Ridge model**.
- **Enhancing the ensemble approach:** We found that individually Ridge gave the best results, then Lasso, then Random Forest, followed by SVR. So we tried the following ensemble approached:
    - We took a **weighted combination of the predictions from SVR, Ridge and Random Forest**, giving higher weight to the better individual performer. And the ensemble of **0.5*ridge + 0.3*rf + 0.2*svr** gave us an amazing performance on the Kaggle leaderboard, securing **Rank 1 and an RMSE of 1,719,022.**
    - We also tried a weighted combination of **Ridge and Lasso, giving equal weight to both.** This gave slightly better results than **Ridge and Lasso as standalones each.**
    - We finally tried a weighted combination of Ridge, Lasso, SVR and Random Forest and the combination **0.3*results_ridge + 0.3*results_lasso + 0.25*results_rf + 0.15*results_svm** did give us **Rank 1 on Kaggle,** but our second lowest **RMSE of 1725979.**

RESULTS:

Date Parsing and Processing	Submission RMSE
Adding number of days open, month and year of opening	1755350
Log treatment on days open, month and year of opening	1750539
Log treatment on days open, dropping month and year	1719022

Transform on Revenue	Submission RMSE
No transform	1762259
Log transform	1773825
Sqrt transform	1719022

City treatment	Submission RMSE
Replace with city counts	1719022
K Means (K=20)	1750803

Type Treatment	**Submission RMSE**
Extra Trees Classifier to replace rary types 'MB' and 'DT'	1750100
KNN Treatment to replace 'MB' which is unaccounted in train data	1719022

P-Variable treatment	**Submission RMSE**
Taking them as continuous	1821659
PCA Processing with 20 Principal Components	1934188
Taking them as categorical	1749479
Adding 'Zeros' feature	1719022

Additional Preprocessing	**Submission RMSE**
Without normalization of numerical attributes	1780995
After normalization of numerical attributes	1719022

Model	**Submission RMSE**
Random Forest	1895861
SVR	1909604
Ensemble of Random Forest and SVR	1782767
Ridge	1749468
Ensemble of Random Forest, Ridge and SVR	1719022
Lasso	1775850
Ensemble of Lasso and Ridge	1744109
Ensemble of Lasso, Ridge, SVR and Random Forest	1725979

SCOPE:

- The solution is **only applicable to Turkish restaurants** and locations on which the data is based. A different location based data may need a different kind of ensemble for accuracy.
- It is limited to only **annual and not seasonal** revenue analysis.
- The accuracy is **limited by the models** we have tried.

CONCLUSIONS:

- When training data is small in size, **the simplest model often gives the best results.** In our case, compared to **SVM and Random Forest** the **ridge** model gave us the best results and heavily reduced our RMSE values.

- Ensembling helps to **average out the error** due to different models and noise in different models. An ensemble of Random Forest, SVR and Ridge gave us the best results among all our attempts, securing **Rank 1 on Kaggle Leaderboard** and an **RMSE value of 1719022.**

REFERENCES:

- https://en.wikipedia.org/wiki/Tikhonov_regularization
- http://www.saedsayad.com/support_vector_machine_reg.htm
- http://kpei.me/blog/wp-content/uploads/2015/05/TFIKaggleReport.pdf
- http://stats.stackexchange.com/questions/17251/what-is-the-lasso-in-regression-analysis
- http://statweb.stanford.edu/~tibs/lasso/simple.html