Homework 4 Part B

10-605/805: Machine Learning with Large Datasets

Due Wednesday, November 1st at 11:59PM Eastern Time

Instructions: There are two parts to this homework, which will have **different deadlines**.

- Part A is due on October 25th and is worth 20% of the grade. No Grace days can be used on this part of the homework.
- Part B (i.e., this document) is due on November 1st and is worth the remaining 80% of the grade.

For 10-805 students: You are required to complete all of the homework for a total point value of 80.

For 10-605 students: You should **not** complete questions 2.1.3 and 2.1.4. The total point value for 10-605 students is 72.

IMPORTANT: Be sure to highlight where your solutions are for each question when submitting to Gradescope otherwise you will be marked 0 and will need to submit a regrade request for every solution unhighlighted in order for fix it!

Note that Homework 4 Part B consists of two parts: this written assignment, and a programming assignment. Remember to fill out the collaboration section found at the end of this homework as per the course policy.

Programming: The programming in this homework is **NOT** autograded however you are required to upload your completed notebooks to Gradescope, otherwise you will not receive credit for the programming sections.

1 Part A: Data Conversion and Preparation

Please complete Homework 4 Part A following its write-up before attempting the coding part below.

2 Part B: Written

2.1 Regression, regression, regression

In this problem you will learn a regression model with the same dataset in several different ways. You may use any mix of manual calculation and computer code that you like. (You should only need short segments of code.) If you use any code, please include your code in the text of your written (part B) submission — not as part of the programming submission.

The dataset for the regressions is below:

x_1	x_2	const	y
1	1	1	1
1	0	1	2
0	1	1	1
2	1	1	-1

For each of the parts 2.1.1–2.1.4 below, you will need to solve a system of linear equations. Please be sure to include the **actual numerical equations** that you solve in the answer to your question, i.e., write out the matrix A and the vector b if you solve $Ax = b$. Hint: the function `numpy.linalg.lstsq` can be very helpful for solving systems of linear equations, including if you need to find the minimum norm solution. But please *do not* use its least squares functionality: i.e., if you want to solve a least squares problem, do it by constructing a set of linear equations that we can solve with zero residual error.

Note that 10-605 students need to solve only three of the five parts below (the first, second, and last), while 10-805 students need to solve all five parts.

2.1.1 Exact linear regression *[4 points]*

Solve the linear regression problem for this dataset exactly using the covariance matrix method. Be sure to report the numerical matrix and vector for the normal equations, as well as the learned weight vector.

Answer:

To solve for w, we need to set up and solve the normal equations:

$$X^T X w = X^T y$$

First, compute $X^T X$:

$$X^T X = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$

Now, compute $X^T y$:

$$X^T y = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 1 \\ -1 \end{bmatrix}$$

Let's compute these using some code.
Alright, let's compute $X^T X$ and $X^T y$ using Python's numpy library.

```
import numpy as np

# Given data
X = np.array([
    [1, 1, 1],
    [1, 0, 1],
    [0, 1, 1],
    [2, 1, 1]
])

y = np.array([1, 2, 1, -1])

# Compute X^T X and X^T y
XTX = np.dot(X.T, X)
XTy = np.dot(X.T, y)

print("XTX:\n", XTX)
print("XTy:\n", XTy)
```

Running this code, we can compute the matrices and vectors we need.
After running the code, the values we get are:

$$X^T X = \begin{bmatrix} 6 & 3 & 4 \\ 3 & 3 & 3 \\ 4 & 3 & 4 \end{bmatrix}$$

and

$$X^T y = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$$

So, our normal equations in matrix form are:

$$\begin{bmatrix} 6 & 3 & 4 \\ 3 & 3 & 3 \\ 4 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$$

Where: w_1 is the weight for x_1, w_2 is the weight for x_2, w_3 is the constant (or bias) term.

Let's solve for the weight vector w using numpy.
To find the weight vector w, we need to solve the equation:

$$X^T X w = X^T y$$

We can find w by multiplying both sides with the inverse of $X^T X$, if it's invertible:

$$w = (X^T X)^{-1} X^T y$$

Using the numpy library, we can compute this:

```
# Solve for w using numpy's linear algebra functionality
w = np.linalg.solve(XTX, XTy)

print("Weight vector w:\n", w)
```

By running the above code, we can determine the values of w_1, w_2, and w_3.
After running the code, we obtain the weight vector w as:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1.66666667 \\ 3 \end{bmatrix}$$

These weights represent the best linear approximation to the given dataset using the exact linear regression method via the covariance matrix. The regression equation is:

$$y = -x_1 - 1.66666667 x_2 + 3$$

This equation provides the best fit line for the given data.

2.1.2 Random projections method *[4 points]*

Solve the same linear regression problem approximately using random projections. Use the fast Johnson-Lindenstrauss method, with the following choices:

- Use one round of fast JL.
- Use the Walsh-Hadamard transform.
- Use random signs (not Gaussians).
- Use a 2d random projection. Recall that we get such a projection by sampling random rows from the transformed matrix or vector.

To make everyone's answers more uniform, please *do not* use truly random numbers. Instead, here are some tables of random numbers; please pick your "random" numbers from these tables in left-to-right order. Use as many as you need; there is no need to use all of them.

Random signs:

$$+1, +1, -1, +1, -1, -1, -1, +1, +1$$

Random indices in $1 \dots 4$:

$$3, 1, 4, 1, 2, 3, 3, 2, 1, 2$$

For reference, the 4×4 WHT matrix is:

$$\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Hint: one step of fast JL applied to a vector u can be written HSu, where H is a WHT matrix and S is a diagonal matrix of random signs. We can then pick out a subset of the entries of the vector HSu corresponding to the projection dimension we want.

Please include in your answer:

- The projection matrix you use.
- The projected dataset (both features and labels).
- The normal equations (in covariance form) for your projected dataset, as a numerical matrix and vector.
- The learned weight vector.

Answer:

Let's proceed with the calculations:

1. Compute the product of Walsh-Hadamard transform H and the diagonal matrix of signs S:

$$HS = H \times S$$

2. Project the dataset: Multiply the data matrix X and vector y with the projection matrix P (selecting the 3rd and 1st rows of HS).

3. Compute the normal equations for the projected dataset.

$$X^T X w = X^T y$$

4. Solve for the weight vector w.

Let's execute these steps in Python:

```
import numpy as np

# Given data
X = np.array([
    [1, 1, 1],
    [1, 0, 1],
    [0, 1, 1],
    [2, 1, 1]
])

y = np.array([1, 2, 1, -1])

# Given WHT matrix
H = 0.5 * np.array([
    [1, 1, 1, 1],
    [1, -1, 1, -1],
    [1, 1, -1, -1],
    [1, -1, -1, 1]
])

# Given signs
S = np.diag([1, 1, -1, 1])

# Compute HS product
HS = np.dot(H, S)

# Select rows based on the given random indices
P = HS[[2, 0], :]

# Project the dataset
X_projected = np.dot(X, P.T)
y_projected = np.dot(y, P.T)

# Compute the normal equations
XTX_projected = np.dot(X_projected.T, X_projected)
XTy_projected = np.dot(X_projected.T, y_projected)

# Solve for w using numpy's linear algebra functionality
w_projected = np.linalg.solve(XTX_projected, XTy_projected)

print("Projection Matrix P:\n", P)
print("\nProjected Dataset (Features):\n", X_projected)
print("\nProjected Dataset (Labels):\n", y_projected)
print("\nXTX for projected dataset:\n", XTX_projected)
print("\nXTy for projected dataset:\n", XTy_projected)
print("\nLearned weight vector for projected dataset:\n", w_projected)
```

Executing the code will provide us the results for the projected dataset, the normal equations, and the learned weight vector.

After executing the code, we obtain the following results: 1. Projection Matrix P:

$$P = \begin{bmatrix} 0.5 & 0.5 & 0.5 & -0.5 \\ 0.5 & 0.5 & -0.5 & 0.5 \end{bmatrix}$$

2. Projected Dataset (Features):

$$X_{\text{projected}} = \begin{bmatrix} 0 & 0.5 & 1 \\ 2 & 0.5 & 1 \end{bmatrix}$$

3. Projected Dataset (Labels):

$$y_{\text{projected}} = \begin{bmatrix} 2.5 \\ 0.5 \end{bmatrix}$$

4. Normal Equations for the Projected Dataset:

$$X_{\text{projected}}^T X_{\text{projected}} = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 0.5 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

$$X_{\text{projected}}^T y_{\text{projected}} = \begin{bmatrix} 1 \\ 1.5 \\ 3 \end{bmatrix}$$

5. Learned Weight Vector for Projected Dataset:

$$w_{\text{projected}} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$$

2.1.3 Kernel version (10-805 only) *[4 points]*

Now switch to using the squared-exponential kernel,

$$k(x, x') = \exp(-\tfrac{1}{2}\|x - x'\|^2)$$

Solve the kernel ridge regression problem exactly using the Gram matrix form, using ridge parameter $\lambda = 1$. Be sure to report the numerical matrix and vector for the regression equations, as well as the resulting example weight vector.

Make a prediction of the label for the following new point: $x_{\text{new}} = (0, 0, 1)$. Report the formula that you use to make the prediction (including numerical values for all quantities) as well as the final prediction.

2.1.4 Random Fourier features (10-805 only) *[4 points]*

Finally, use random Fourier features to approximate the same kernel ridge regression problem as in the previous part. As before, please *do not* use truly random numbers. Instead, use the table of "random" Gaussian samples below. Use just one random feature; this is obviously too few for a real problem, but will make calculation shorter.

Random Gaussian samples:

$$0.5, -1.2, -0.7, 0.3, 1.9, 2.1, -0.1$$

Be sure to report the numerical matrix and vector for the normal equations that you solve, as well as the resulting weight vector. Finally, make a prediction of the label for x_{new} (the same point as in the previous part). Report the formula that you use to make the prediction (including numerical values for all quantities) as well as the final prediction.

Hint: you can do complex arithmetic in Python. A complex number is something like `2+3j`. You can use `numpy.exp(2j)` for the complex exponential, and you can use `x.conj() @ y` for complex dot product. Note that Python uses j instead of i for the imaginary unit, and that you have to type something like `1j` instead of `j` to disambiguate an imaginary number from a variable named `j`.

2.1.5 When and why? *[4 points]*

For each of the methods that you used above (exact covariance-form regression, random projection for covariance form, exact kernel ridge regression, and random Fourier features for kernel ridge regression), please give a few sentences saying when we might use this method and why. (10-605 students: please answer for all four methods, even though you only solved two of them.)

Answer:

1. **Exact Covariance-Form Regression**:
   - *When to use*: This method is appropriate for datasets that aren't prohibitively large and where the relationship between predictors and the response variable can be assumed to be linear.
   - *Why*: It provides an exact solution, ensuring the best linear fit for the given data. It's computationally efficient for smaller datasets and provides a clear interpretability of the relation between predictors and response.
2. **Random Projection for Covariance Form**:
   - *When to use*: This method is suitable for very high-dimensional datasets where exact methods would be computationally expensive or infeasible.
   - *Why*: Random projections can significantly reduce dimensionality while approximately preserving pairwise distances. It offers a trade-off: we sacrifice some accuracy for a considerable gain in computational efficiency.
3. **Exact Kernel Ridge Regression**:
   - *When to use*: When the relationship between predictors and response is non-linear and the dataset isn't too large. Also, when we have some knowledge or intuition about the nature of the underlying non-linearity which allows for appropriate kernel selection.
   - *Why*: Kernel methods map the original data into a higher-dimensional space where it becomes linearly separable. Ridge regression, specifically, adds regularization which can help prevent over-fitting, especially when the number of features is high.
4. **Random Fourier Features for Kernel Ridge Regression**:
   - *When to use*: For large-scale datasets where exact kernel methods would be too computationally intensive. It's especially useful when we want to employ kernel methods (due to non-linearity in the data) but need a scalable solution.
   - *Why*: Random Fourier features allow an approximation of the kernel trick without computing the full kernel matrix, thus making kernel methods scalable. By mapping the data into a randomized low-dimensional feature space, we can use linear algorithms (like ridge regression) efficiently, while still capturing non-linear patterns.

3 Part B: Programming

In this part of the homework, you will perform exploratory data analysis (EDA) and data cleaning, and then train models with the original features. You will then perform feature engineering similar to what we did in Homework 1 (TF-IDF and Bag-of-Words), and then train models with these new features.

3.1 Setting up EMR and Spark

With our data ready in S3, it's now time to configure and create an EMR (Elastic MapReduce) cluster and run Spark, starting from the notebook `hw4.ipynb`. Include at least Hadoop, JupyterHub, and Spark in your cluster software configuration. Set `maximizeResourceAllocation` to true (see here) in software settings. Select your EC2 key pair.

Then, log in to jupyter (learn about login credentials here), upload your notebook, and start working!

Again, cost management is key. Because we might be running a cluster of machines, this could easily blow up your budget. We recommend using at most 1 Driver and 1 Core of type `m5.xlarge` while developing and debugging on a subset of MSD. You could scale this up to multiple Core workers when doing the final run. You may also utilize AWS spot instances to save cost during development.

Note that, although EMR is made up of EC2 instances, unlike EC2, you cannot stop an EMR cluster—you can only terminate it. You should plan your strategy accordingly. **Do not forget to download your code** before you terminate a cluster when you are done.

3.2 Preprocessing

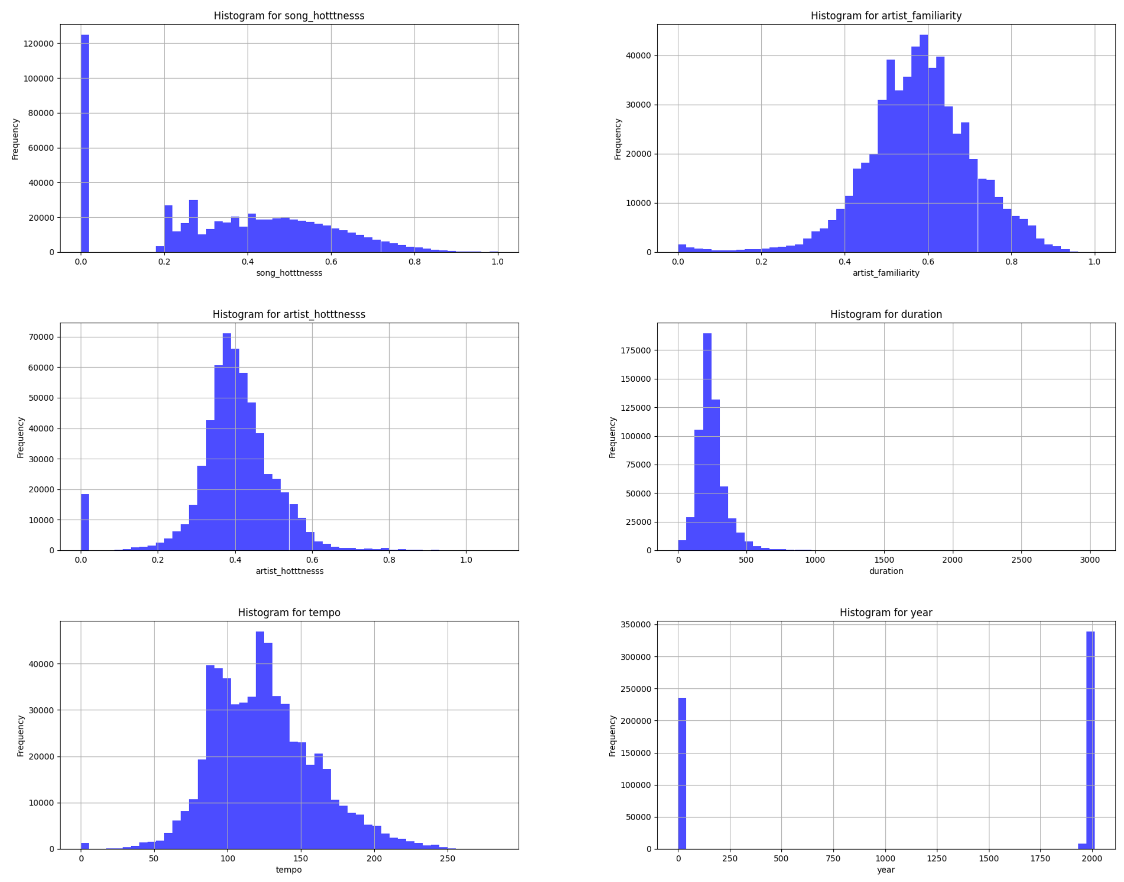
Open JuptyerHub on your EMR cluster, similar to what you did on HW3, and upload your `hw4.ipynb` to begin working on it!

3.3 Exploratory Data Analysis *[11 points]*

(a) *[1 points]* Explain why the two features seem problematic (after performing .summary() operation).

The features 'danceability' and 'energy' are problematic as they only contain a single value, 0.0, for all entries in the dataset. The lack of variability makes them non-informative and potentially redundant for further analysis or modeling.

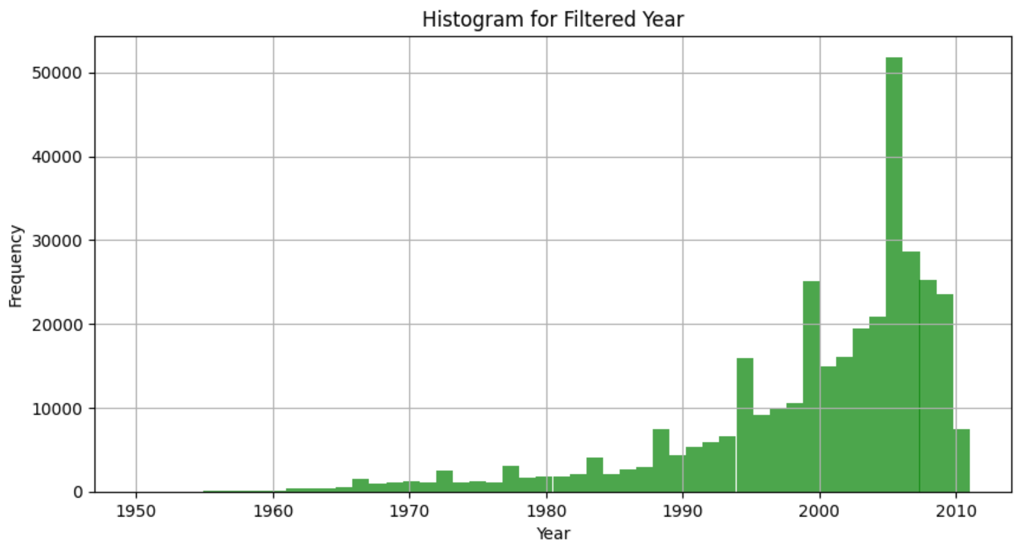
(b) *[3 points]* Histograms (remember to label them)



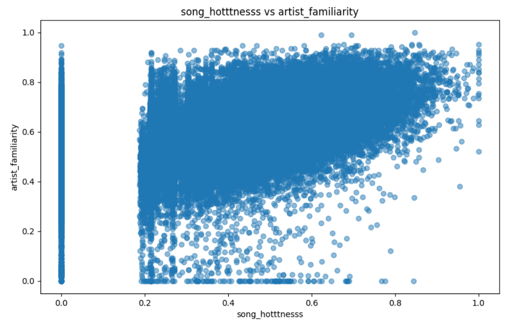
(c) *[1 points]* Explain what is strange about `year`'s distribution and what might cause this. Describe how you could filter `year` to make its histogram look more balanced.

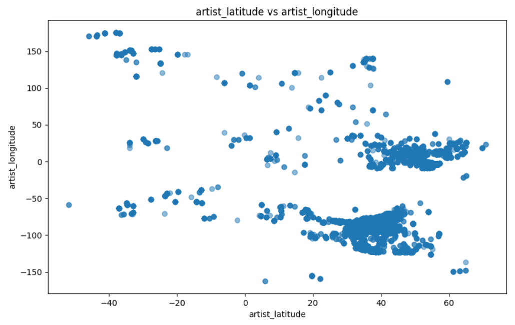
The year's distribution contains high count for the year 0-50. Over 200,000 entries are recorded for this year, which stands out as an anomaly, especially if we consider that there is no year "0" in our common Gregorian calendar. It could be placeholder for missing data where a common practice in data handling is to use a placeholder value, like "0", when the actual data is missing or unknown. It's possible that many records lacked the year information, and "0" was inputted as a default value. We can set a threshold to only include year higher than 1950.

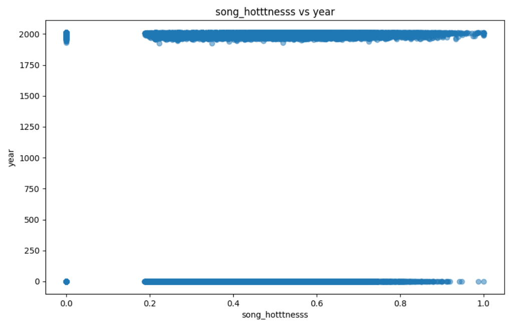
(d) *[1 points]* New histogram for `year`.



(e) *[3 points]* Provide plots for the three pairs. Describe your findings.







For song hotttnesss and artist familiarity scatter plot, when song hotttnesss is above 0.2, we can observe a slightly positive relationship between song hotttnesss and artist familiarity. For artist latitude and artist longitude scatter plot, there do not show any pattern. However, it also show artists come from various parts of the world with a strong clustering in specific regions. For song hotttnesss and year scatter plot, I do not observe any pattern. It is just that when year is 0, we have records on song hotttnesss.

(f) *[2 points]* Think about what simple technique you could use to visualize large datasets while retaining a similar data distribution. Briefly describe what you did.

What I did is sampling. I randomly sample a fraction of my data to reduce the number of points I am plotting. This can often give me a good overview of the data distribution without overwhelming the plotting tool.

3.4 Data Cleaning *[8 points]*

(a) *[1 points]* Your justification for dropping the two features.

From a statistical perspective, a feature with no variance will not contribute to any model's predictive capability. This is especially true in models like linear regression where feature variance is crucial. Also, removing such features will reduce the dataset's dimensionality, which in turn can speed up certain computations and processes, especially when handling big data frameworks like Spark.

(b) *[2 points]* Compare the two numbers and explain the advantages and potential problem of doing this step. What other techniques could you use to potentially do better?

After dropping records with year values less than or equal to 1920, we have gone from an initial count of 581,965 samples to 346,444 samples. This is a reduction of 235,521 records, which is approximately 40.5 percent of the original dataset.
We can consider imputation to potentially do better: Instead of dropping older records, we can impute missing or problematic values using methods like mean, median, or mode imputation, regression imputation, or even more sophisticated methods like KNN imputation.
We can also consider feature engineering: Use the year as a feature and bucketize it. For example, make categories like "pre-1920", "1920-1950", and so on, to capture the trends of each era without dropping data.

(c) *[1 points]* State the two features.

artist latitude and artist longitude

(d) *[2 points]* Explain your proposed solution and discuss its pros and cons.

First one is geocoding from artist information: If the dataset has other location-related attributes (e.g., artist's hometown or country), a geocoding API could be used to obtain missing latitude and longitude values. Pros: Retains more data and fills in the missing values with potentially accurate data. Cons: Relies on third-party services, might be time-consuming and can have API call limits. Accuracy is also dependent on the quality of other location-related attributes.
Second one is imputation using K-Nearest Neighbors (KNN): For missing latitude and longitude, impute using KNN where nearby or similar artists' locations inform the missing value. Pros: Uses the structure in the data to predict missing values, might give more accurate location estimates. Cons: Computationally intensive, especially for large datasets. Imputed values are based on data structure rather than real-world location logic.

(e) *[2 points]* Report the percentage:

21.83 percent

3.5 Baseline *[13 points]*

(a) *[2 points]* Explain why treating this as a classification problem might be a sensible choice.

Classification can make the problem more interpretable. By dividing songs into ”popular” and ”not popular” based on a threshold (like the average song hotttnesss value), we can make clear-cut decisions and recommendations. If the end goal is to decide, for example, whether to add a song to a ”recommended” playlist or not, a binary classification provides a straightforward approach.

(b) *[1 points]* Report what percentage of songs are assigned the “popular” label.

55.99 percent

(c) *[1 points]* Explain why we shift the year.

Many machine learning algorithms, especially those relying on distances (like k-NN) or gradient descent optimization (like neural networks), work better when the input features are on a similar scale. By shifting years, we are ensuring the values for the ’year’ feature are closer in range to other normalized or scaled features in the dataset.

(d) *[2 points]* Explain what scaling means and why we want to perform scaling before the learning step.

Scaling refers to the process of transforming the range of the feature values, usually to a standard range like [0, 1] or to have a mean of 0 and a standard deviation of 1.
Features with larger scales might dominate the objective function in certain algorithms, potentially leading to suboptimal solutions. Scaling ensures that each feature has an equal initial influence. In certain models, like linear regression, scaling can help in better interpreting the feature coefficients, as they would be on a comparable scale.

(e) *[3 points]* Explain the difference between these two metrics and when AUC might be more useful than accuracy.

Accuracy is a direct metric that measures the proportion of correct predictions in the total predictions made. Of all the classifications, it tells how many were actually correct. AUC stands for ”Area Under the Receiver Operating Characteristic Curve.” It provides an aggregate measure of model performance across all possible classification thresholds. AUC is more useful than accuracy, especially when the classes are imbalanced, when the cost of false positives and false negatives are very different, and when we want to evaluate the model’s performance irrespective of the threshold.

(f) *[4 points]* Calculate the train and test AUC of both models and report them.

Models	Train AUC	Test AUC
Logistic Regression	0.7545358768092437	0.7552175140382875
Random Forest	0.7636757605336973	0.7634130041506864

3.6 Featurization: Bag-of-Words and TF-IDF *[8 points]*

(a) *[3 points]* Explain what the `vocabSize` hyperparameter means in the context of Bag-of-Words.

The vocabSize hyperparameter in the context of Bag-of-Words (BoW) determines the maximum size of the vocabulary, effectively setting a limit on the number of unique tokens or words that are considered when converting text data into numerical vectors. A smaller vocabSize results in simpler, lower-dimensional feature vectors, which can help prevent overfitting but might omit potentially important information. A larger vocabSize captures more unique words but increases the dimensionality of the feature vectors, potentially leading to sparser vectors and requiring more computational resources.

(b) *[3 points]* Other than featurizing texts, what other feature engineering would you do on the dataset? Briefly describe one.

We have artist latitude and artist longitude as features. We could transform these into a more meaningful representation, like calculating the distance of the artist's location from a specific music hub (e.g., Nashville, Los Angeles, or London). This could potentially capture the influence of geographical location on a song's popularity or an artist's success.

(c) *[2 points]* Explain where this number "31" comes from.

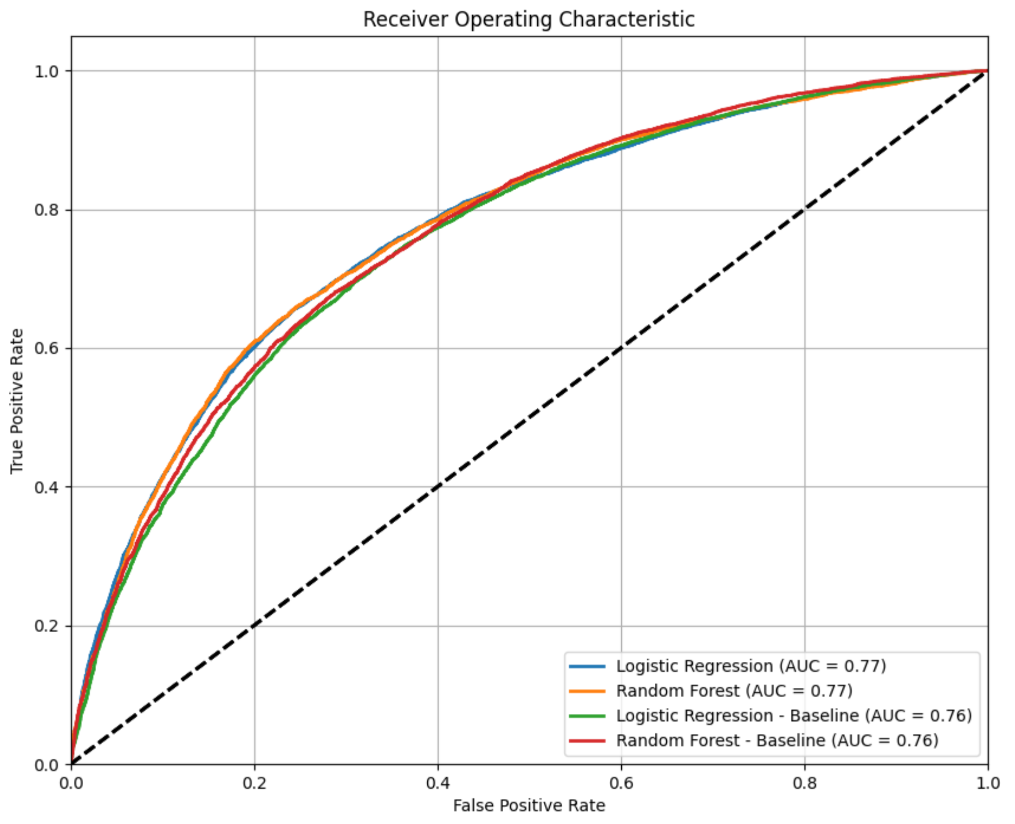
The number 31 comes from 16 original numeric features, 5 features from TF-IDF (as we set TF-IDF-NUM-FEATS to 5), and 10 features from BoW (as we set BOW-VOCAB-SIZE to 10), totaling 31 features

3.7 Modeling with New Features *[10 points]*

(a) *[4 points]* Evaluate train and test AUC for each model and report them.

Models	Train AUC	Test AUC
Logistic Regression	0.7673455432649633	0.7681461257531581
Random Forest	0.7695213355843427	0.7693112770104944

(b) *[6 points]* Include the plot and your explanations.



This is ROC curve for both logistic regression and random forest, including their baseline performance. As we can see from plot. Adding new features slightly improve the curve. Especially when false positive rate is 0.2, we have higher true positive rate for models with new feature.

3.8 Do Your Best *[7 points]*

(a) *[2 points]* Your final AUC:.

Train AUC: 0.9994036370497007
Test AUC: 0.8620067672340108

(b) *[3 points]* Your model and hyperparameters.

Model: Random Forest

Hyperparameters:
numTrees: 30
maxDepth: 20
minInstancesPerNode: 1
impurity: gini
maxBins: 32
bootstrap: True
cacheNodeIds: False
checkpointInterval: 10
featureSubsetStrategy: auto

(c) *[2 points]* Describe your approach.

Previously, we can observe that random forest performs better than logistic regression on this dataset. Therefore, in this part, I still use random forest since random forest is a powerful ensemble learning method, and tuning its hyperparameters can lead to improved model performance. I set up a parameter grid for hyperparameter tuning. The example grid includes number of trees (numTrees), maximum depth of the trees (maxDepth), and minimum instances per node (minInstancesPerNode). I also perform 5-fold cross-validation using CrossValidator. After fitting the model, I evaluate its performance on both the training and test datasets and print out the AUC values.

3.9 Reflection

[3 points] What challenges did you face in HW4 Section 3? How did you overcome these challenges? What did you learn from HW4 ?"

In HW4 Section 3, I grappled with the complexities of handling an extensive dataset, which presented significant computational challenges and prolonged processing times. In the last part, I delved into various machine learning models, encountering the intricacies of feature engineering and the criticality of hyperparameter tuning to enhance model accuracy. The extensive waiting times for model training and tuning underscored the need for a strategic approach, leading me to employ sampling techniques and simplified models for preliminary experimentation. This hands-on experience was invaluable, deepening my understanding of big data technologies, machine learning best practices, and the art of making informed trade-offs between computational efficiency and model performance. Ultimately, HW4 was a profound learning journey, equipping me with practical skills and insights that extended beyond the theoretical knowledge into the realm of applying machine learning at scale.

4 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   No

   (b) If you answered ‘yes’, give full details (e.g. “Jane Doe explained to me what is asked in Question 3.4”)

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   No

   (b) If you answered ‘yes’, give full details (e.g. “I pointed Joe Smith to section 2.3 since he didn’t know how to proceed with Question 2”)

3. (a) Did you find or come across code that implements any part of this assignment?

   No

   (b) If you answered ‘yes’, give full details (book & page, URL & location within the page, etc.).