CPSC 340 Assignment 3 (due Friday, Feb 6 at 11:55pm)

Student name: José Abraham Torres Juárez
Student id: 79507828

Instructions

Rubric: {mechanics:5}

IMPORTANT!!! Before proceeding, please carefully read the general homework instructions at `https://www.cs.ubc.ca/~fwood/CS340/homework/`. The above 5 points are for following the submission instructions. You can ignore the words "mechanics", "reasoning", etc.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

1 Finding Similar Items

For this question we'll use the Amazon product data set[1] from `http://jmcauley.ucsd.edu/data/amazon/`. We will focus on the "Patio, Lawn, and Garden" section. You should start by downloading the ratings at `https://stanford.io/2Q7QTvu` and place the file in your `data` directory with the original filename. Once you do that, running `python main.py -q 1` should do the following steps:

- Load the raw ratings data set into a Pandas dataframe.
- Construct the user-product matrix as a sparse matrix (to be precise, a `scipy.sparse.csr_matrix`).
- Create bi-directional mappings from the user ID (e.g. "A2VNYWOPJ13AFP") to the integer index into the rows of `X`.
- Create bi-directional mappings from the item ID (e.g. "0981850006") to the integer index into the columns of `X`.

1.1 Exploratory data analysis

1.1.1 Most popular item

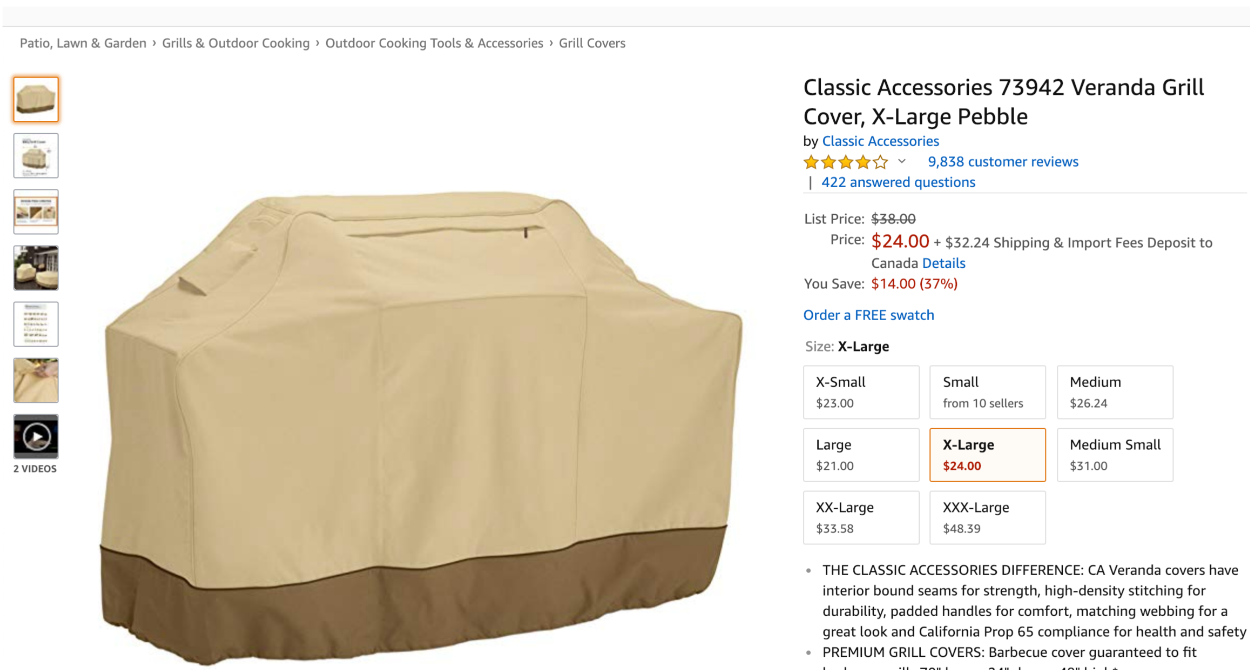
Rubric: {code:1}

Find the item with the most total stars. Submit the product name and the number of stars.

[1]The author of the data set has asked for the following citations: (1) Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. R. He, J. McAuley. WWW, 2016, and (2) Image-based recommendations on styles and substitutes. J. McAuley, C. Targett, J. Shi, A. van den Hengel. SIGIR, 2015.

Note: once you find the ID of the item, you can look up the name by going to the url `https://www.amazon.com/dp/ITEM_ID`, where `ITEM_ID` is the ID of the item. For example, the URL for item ID "B00CFM0P7Y" is `https://www.amazon.com/dp/B00CFM0P7Y`.

ID: B000HCLLMM
Name: Classic Accessories 73942 Veranda Grill Cover, X-Large Pebble
Stars: 14454.0
Link: `https://www.amazon.com/dp/B000HCLLMM`



1.1.2 User with most reviews

Rubric: {code:1}

Find the user who has rated the most items, and the number of items they rated.

ID: A100WO06OQR8BQ
Items rated: 161

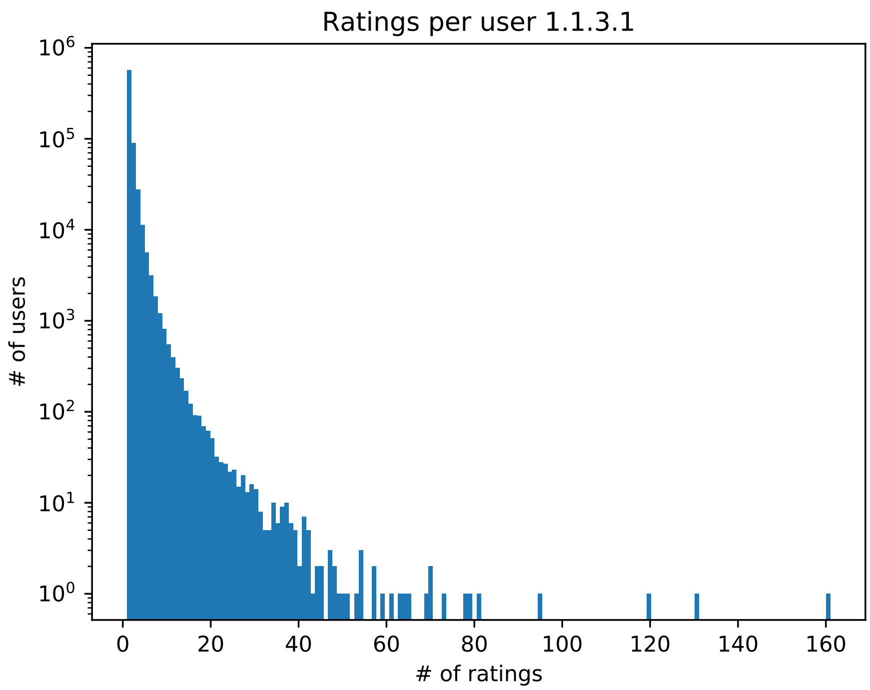
1.1.3 Histograms

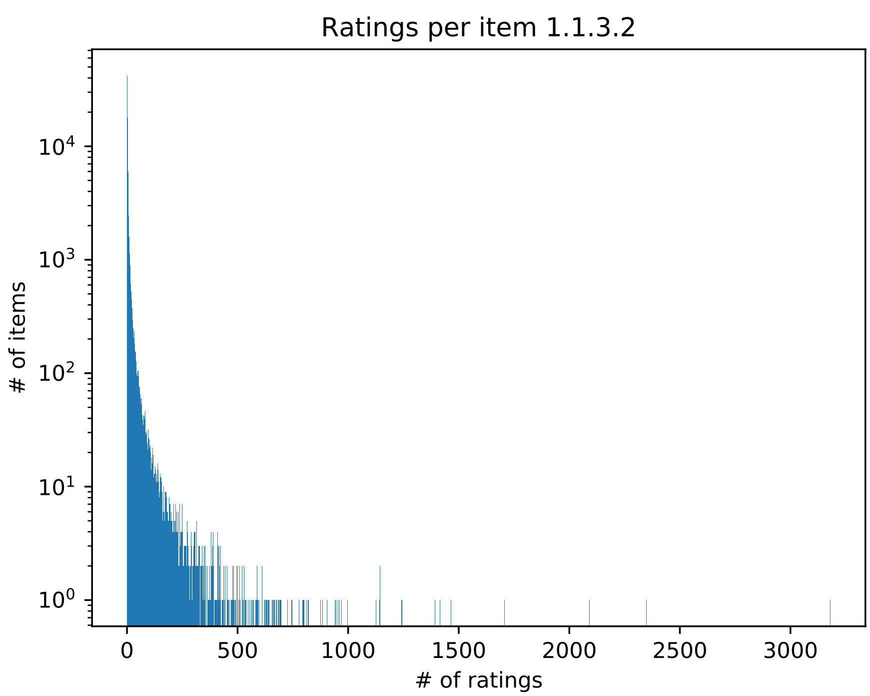
Rubric: {code:2}

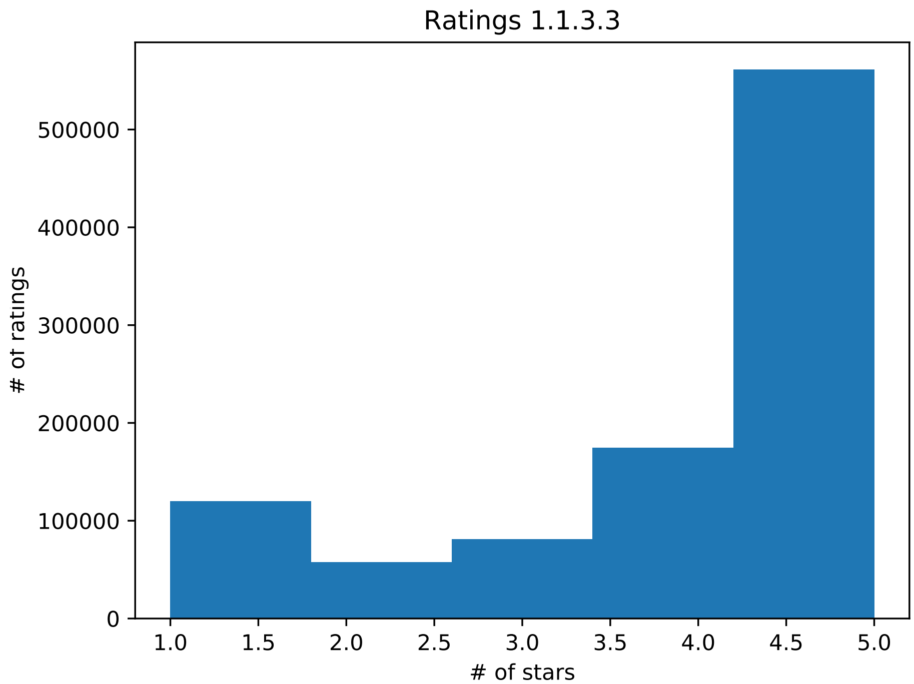
Make the following histograms:

1. The number of ratings per user
2. The number of ratings per item
3. The ratings themselves

Note: for the first two, use `plt.yscale('log', nonposy='clip')` to put the histograms on a log-scale. Also, you can use `X.getnnz` to get the total number of nonzero elements along a specific axis.







1.2 Finding similar items with nearest neighbours

Rubric: {code:6}

We'll use scikit-learn's `neighbors.NearestNeighbors` object to find the items most similar to the example item above, namely the Brass Grill Brush 18 Inch Heavy Duty and Extra Strong, Solid Oak Handle, at URL `https://www.amazon.com/dp/B00CFM0P7Y`.

Find the 5 most similar items to the Grill Brush using the following metrics:

1. Euclidean distance (the `NearestNeighbors` default)
2. Normalized Euclidean distance (you'll need to do the normalization)
3. Cosine similarity (by setting `metric='cosine'`)

Some notes/hints...

- If you run `python main.py -q 1.2`, it will grab the row of `X` associated with the grill brush. The mappers take care of going back and forther between the IDs (like "B00CFM0P7Y") and the indices of the sparse array $(0, 1, 2, \ldots)$.
- Keep in mind that scikit-learn's `NearestNeighbors` is for taking neighbors across rows, but here we're working across columns.
- Keep in mind that scikit-learn's `NearestNeighbors` will include the query item itself as one of the nearest neighbours if the query item is in the "training set".
- Normalizing the columns of a matrix would usually be reasonable to implement, but because X is stored as a sparse matrix it's a bit more of a mess. Therefore, use `sklearn.preprocessing.normalize` to help you with the normalization in part 2.

Did normalized Euclidean distance and cosine similarity yields the same similar items, as expected?

1. Euclidean distance

ID	Euclidean Distance
B00IJB4MLA	74.242
B00IJ8951K	75.146
B00HVXLFZS	76.517
B00EXDNA3O	76.517
B00743MZCM	76.517

2. Normalized Euclidean distance

ID	Normalized Euclidean Distance
B00IJB4MLA	76.385
B00IJB5MCS	76.408
B00IJ8951K	76.436
B00EF4206E	76.476
B00EF0I5Q2	76.568

3. Cosine similarity

ID	Cosine Similarity
B00IJB4MLA	0.698
B00IJB5MCS	0.721
B00IJ8951K	0.749
B00EF4206E	0.789
B00EF0I5Q2	0.881

As we can see in the tables above the cosine similarity yields the same similar items of the Normalized Euclidean Distance. There are just two products in common of the three methods, the most similar item in all methods B00IJB4MLA, and the 2nd (Euclidean Distance) or 3rd (Normalized, Cosine) item B00IJ8951K

1.3 Total popularity

Rubric: {reasoning:2}

For both Euclidean distance and cosine similarity, find the number of reviews for each of the 5 recommended items and report it. Do the results make sense given what we discussed in class about Euclidean distance vs. cosine similarity and popular items?

Note: in `main.py` you are welcome to combine this code with your code from the previous part, so that you don't have to copy/paste all that code in another section of `main.py`.

1. Euclidean distance

ID	# Reviews
B00IJB4MLA	45
B00IJ8951K	1
B00HVXLFZS	6
B00EXDNA3O	22
B00743MZCM	1

2. Cosine similarity

ID	# Reviews
B00IJB4MLA	45
B00IJB5MCS	55
B00IJ8951K	1
B00EF4206E	1
B00EF0I5Q2	2

2 Matrix Notation and Minimizing Quadratics

2.1 Converting to Matrix/Vector/Norm Notation

Rubric: {reasoning:3}

Using our standard supervised learning notation (X, y, w) express the following functions in terms of vectors, matrices, and norms (there should be no summations or maximums).

1. $\max_{i\in\{1,2,\ldots,n\}} |w^T x_i - y_i|$.

$$||Xw - y||_\infty$$

2. $\sum_{i=1}^n v_i(w^T x_i - y_i)^2 + \frac{\lambda}{2}\sum_{j=1}^d w_j^2$.

let V be a $n \times n$ diagonal matrix with v_i along the diagonal

$$||V(Xw - y)^2||_1 + \frac{\lambda}{2}||w||_2^2$$

3. $\left(\sum_{i=1}^n |w^T x_i - y_i|\right)^2 + \frac{1}{2}\sum_{j=1}^d \lambda_j |w_j|$.

let Λ be a $d \times d$ diagonal matrix with λ_i along the diagonal

$$||Xw - y||_1^2 + \frac{1}{2}||\Lambda w||_1$$

Note that in part 2 we give a *weight* v_i to each training example and the value λ is a non-negative scalar, whereas in part 3 we are regularizing the parameters with different weights λ_j. You can use V to denote a diagonal matrix that has the values v_i along the diagonal, and Λ as a diagonal matrix that has the λ_j values along the diagonal. You can assume that all the v_i and λ_i values are non-negative.

2.2 Minimizing Quadratic Functions as Linear Systems

Rubric: {reasoning:3}

Write finding a minimizer w of the functions below as a system of linear equations (using vector/matrix notation and simplifying as much as possible). Note that all the functions below are convex so finding a w with $\nabla f(w) = 0$ is sufficient to minimize the functions (but show your work in getting to this point).

1. $f(w) = \frac{1}{2}\|w - v\|^2$ (projection of v onto real space).

$$f(w) = \frac{1}{2}(w^T w - 2w^T v + v^T v)$$

$$f(w) = \frac{1}{2}w^T w - w^T v + \frac{1}{2}v^T v$$

$$\nabla f(w) = w - v + 0$$

$$0 = w - v$$

$$w = v$$

2. $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^T\Lambda w$ (least squares with weighted regularization).

$$f(w) = \frac{1}{2}(w^TX^TXw - 2w^TX^Ty + y^Ty) + \frac{1}{2}w^T\Lambda w$$

let $A = X^TX$
let $B = X^Ty$
let $C = y^Ty$

$$f(w) = \frac{1}{2}w^TAw - w^TB + C) + \frac{1}{2}w^T\Lambda w$$
$$\nabla f(w) = Aw - B + 0 + \Lambda w$$
$$0 = Aw + \Lambda w - B$$
$$(A + \Lambda)w = B$$
$$(X^TX + \Lambda)w = X^Ty$$

3. $f(w) = \frac{1}{2}\sum_{i=1}^n v_i(w^Tx_i - y_i)^2 + \frac{\lambda}{2}\|w - w^0\|^2$ (weighted least squares shrunk towards non-zero w^0).

$$f(w) = \frac{1}{2}||(Xw - y)^TV(Xw - y)||_1 + \frac{\lambda}{2}\|w - w^0\|^2$$
$$f(w) = \frac{1}{2}||(V^TXw - V^Ty)^T(Xw - y)||_1 + \frac{\lambda}{2}\|w - w^0\|^2$$
$$f(w) = \frac{1}{2}||w^TX^TV^TXw - 2w^TX^TV^Ty + y^TV^Ty|| + \frac{\lambda}{2}(w^Tw - 2w^Tw^0 + w^{0T}w^0)$$

let $A = X^TV^TX$
let $B = X^TV^Ty$
let $C = y^TV^Ty$

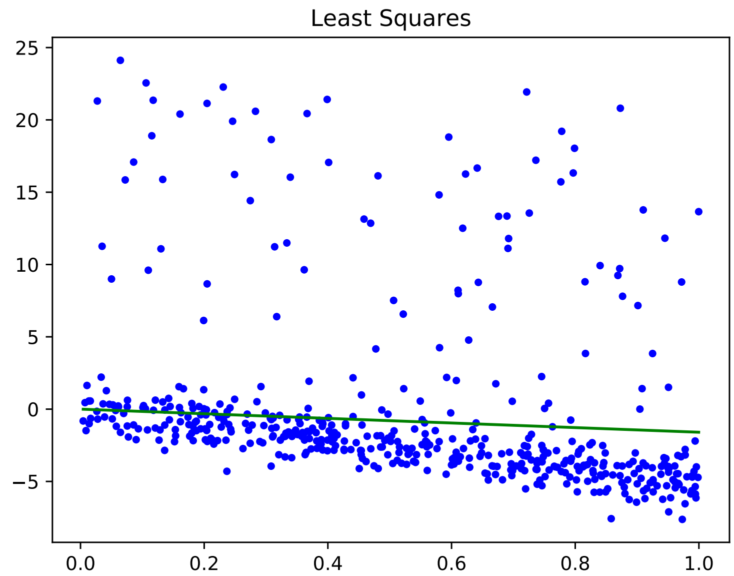
$$f(w) = \frac{1}{2}||w^TAw - 2w^TB + C|| + \frac{\lambda}{2}||w^Tw - 2w^Tw^0 + w^{0T}w^0||$$
$$f(w) = ||\frac{1}{2}w^TAw - w^TB + \frac{1}{2}C|| + ||\frac{\lambda}{2}w^Tw - \lambda w^Tw^0 + \frac{\lambda}{2}w^{0T}w^0||$$
$$\nabla f(w) = Aw - B + 0 + \lambda w - \lambda w^0 + 0$$
$$0 = Aw + \lambda w - B - \lambda w^0$$
$$Aw + \lambda w = B - \lambda w^0$$
$$(A + \lambda I)w = B + \lambda w^0$$
$$(X^TV^TX + \lambda I)w = X^TV^Ty + \lambda w^0$$

Above we assume that v and w^0 are d by 1 vectors (in part 3 v is a vector of length n by 1), and Λ is a d by d diagonal matrix (with positive entries along the diagonal). You can use V as a diagonal matrix containing the v_i values along the diagonal.

Hint: Once you convert to vector/matrix notation, you can use the results from class to quickly compute these quantities term-wise. As a sanity check for your derivation, make sure that your results have the right dimensions. As a sanity check, make that the dimensions match for all quantities/operations: in order to make the dimensions match for some parts you may need to introduce an identity matrix. For example, $X^TXw + \lambda w$ can be re-written as $(X^TX + \lambda I)w$.

3 Robust Regression and Gradient Descent

If you run `python main.py -q 3`, it will load a one-dimensional regression dataset that has a non-trivial number of 'outlier' data points. These points do not fit the general trend of the rest of the data, and pull the least squares model away from the main downward trend that most data points exhibit:



Note: we are fitting the regression without an intercept here, just for simplicity of the homework question. In reality one would rarely do this. But here it's OK because the "true" line passes through the origin (by design). In Q4.1 we'll address this explicitly.

3.1 Weighted Least Squares in One Dimension

Rubric: {code:3}

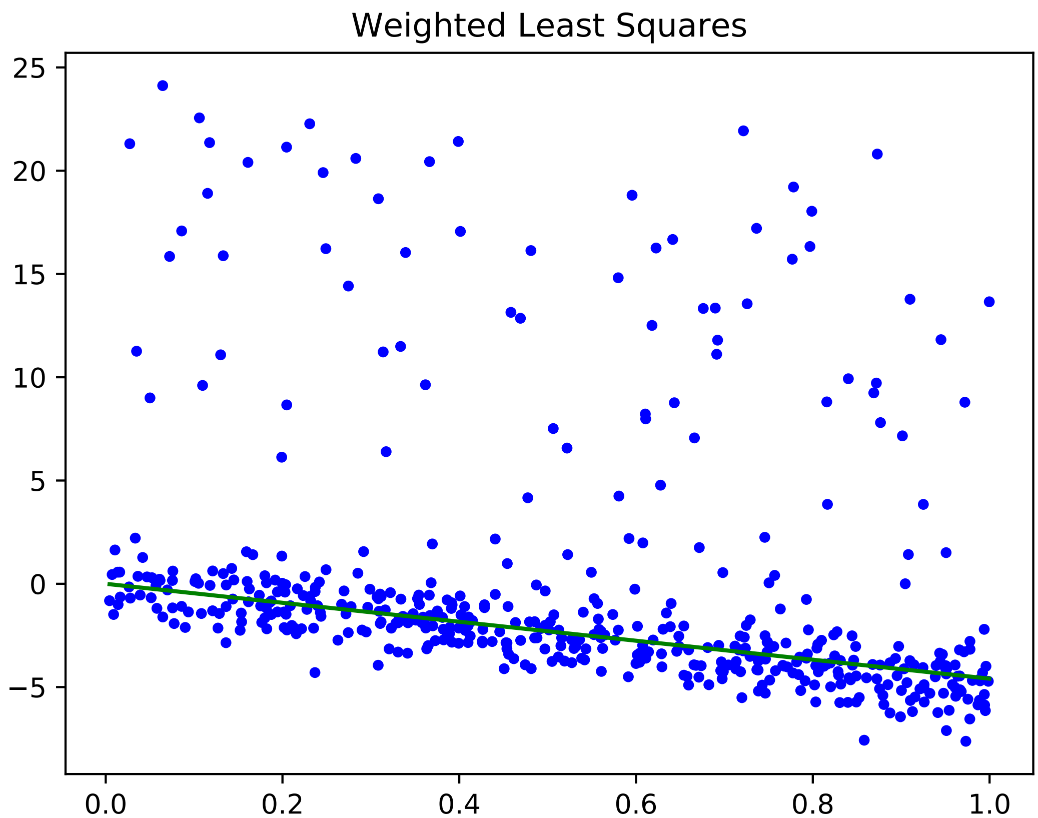
One of the most common variations on least squares is *weighted* least squares. In this formulation, we have a weight v_i for every training example. To fit the model, we minimize the weighted squared error,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2.$$

In this formulation, the model focuses on making the error small for examples i where v_i is high. Similarly, if v_i is low then the model allows a larger error. Note: these weights v_i (one per training example) are completely different from the model parameters w_j (one per feature), which, confusingly, we sometimes also call "weights".

Complete the model class, `WeightedLeastSquares`, that implements this model (note that Q2.2.3 asks you to show how a few similar formulation can be solved as a linear system). Apply this model to the data containing outliers, setting $v = 1$ for the first 400 data points and $v = 0.1$ for the last 100 data points (which are the outliers). Hand in your code and the updated plot.

```
# Least squares where each sample point X has a weight associated with it.
# inherits the predict() function from LeastSquares
class WeightedLeastSquares(LeastSquares):
    def fit(self,X,y,z):
        # X^T z^T X w = X^T z^T y
        self.w = solve(X.T@z.T@X, X.T@z.T@y)
```



3.2 Smooth Approximation to the L1-Norm

Rubric: {reasoning:3}

Unfortunately, we typically do not know the identities of the outliers. In situations where we suspect that there are outliers, but we do not know which examples are outliers, it makes sense to use a loss function that is more robust to outliers. In class, we discussed using the sum of absolute values objective,

$$f(w) = \sum_{i=1}^{n} |w^T x_i - y_i|.$$

This is less sensitive to outliers than least squares, but it is non-differentiable and harder to optimize. Nevertheless, there are various smooth approximations to the absolute value function that are easy to optimize. One possible approximation is to use the log-sum-exp approximation of the max function[2]:

$$|r| = \max\{r, -r\} \approx \log(\exp(r) + \exp(-r)).$$

Using this approximation, we obtain an objective of the form

$$f(w) = \sum_{i=1}^{n} \log\left(\exp(w^T x_i - y_i) + \exp(y_i - w^T x_i)\right).$$

which is smooth but less sensitive to outliers than the squared error. Derive the gradient ∇f of this function with respect to w. You should show your work but you do not have to express the final result in matrix notation.

$$\nabla f(w) = \frac{x_i \exp(w^T x_i - y_i) - x_i \exp(y_i - w^T x_i)}{\exp(w^T x_i - y_i) + \exp(y_i - w^T x_i)}$$

3.3 Robust Regression

Rubric: {code:3}

The class `LinearModelGradient` is the same as `LeastSquares`, except that it fits the least squares model using a gradient descent method. If you run `python main.py -q 3.3` you'll see it produces the same fit as we obtained using the normal equations.

The typical input to a gradient method is a function that, given w, returns $f(w)$ and $\nabla f(w)$. See `funObj` in `LinearModelGradient` for an example. Note that the `fit` function of `LinearModelGradient` also has a numerical check that the gradient code is approximately correct, since implementing gradients is often error-prone.[3]

An advantage of gradient-based strategies is that they are able to solve problems that do not have closed-form solutions, such as the formulation from the previous section. The class `LinearModelGradient` has most of the implementation of a gradient-based strategy for fitting the robust regression model under the log-sum-exp approximation. The only part missing is the function and gradient calculation inside the `funObj` code. Modify `funObj` to implement the objective function and gradient based on the smooth approximation to the absolute value function (from the previous section). Hand in your code, as well as the plot obtained using this robust regression approach.

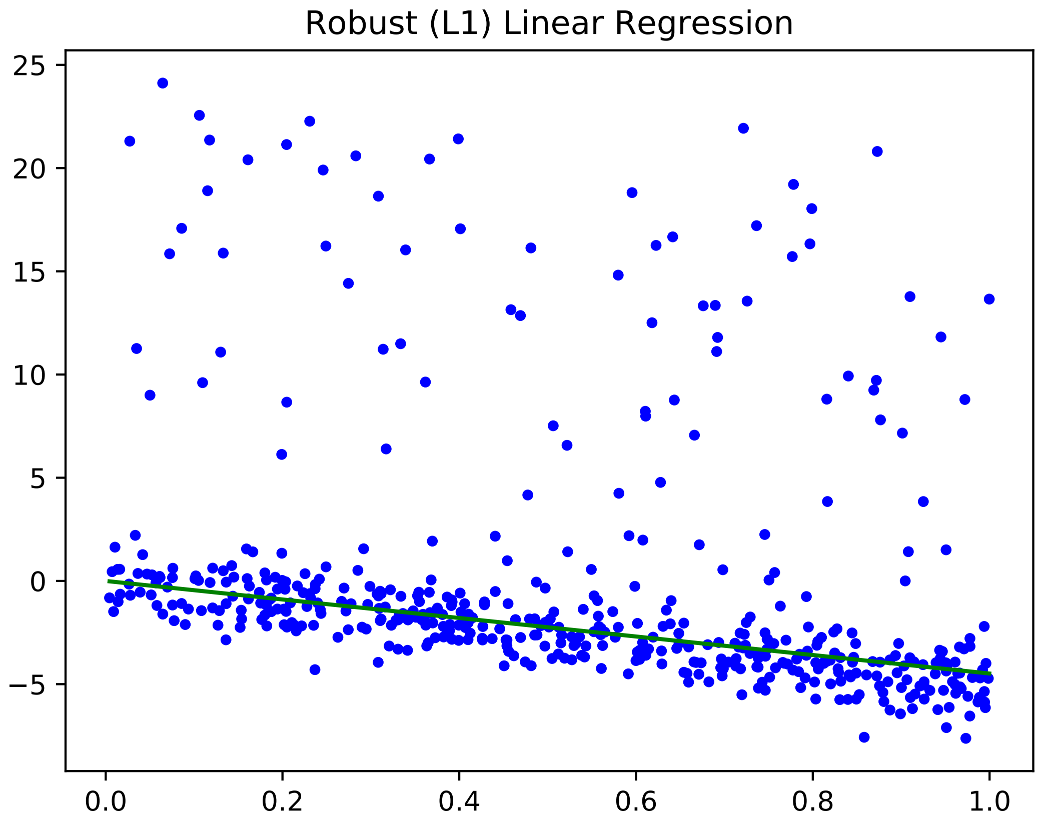
```
def funObj(self,w,X,y):
    # Calculate the function value
    f = np.sum(np.log(np.exp(X@w - y) + np.exp(y - X@w)))

    # Calculate the gradient value
    g = X.T@((np.exp(X@w - y) - np.exp(y - X@w))/(np.exp(X@w - y) + np.exp(y - X@w)))

    return (f,g)
```

[2]Other possibilities are the Huber loss, or $|r| \approx \sqrt{r^2 + \epsilon}$ for some small ϵ.

[3]Sometimes the numerical gradient checker itself can be wrong. See CPSC 303 for a lot more on numerical differentiation.



4 Linear Regression and Nonlinear Bases

In class we discussed fitting a linear regression model by minimizing the squared error. In this question, you will start with a data set where least squares performs poorly. You will then explore how adding a bias variable and using nonlinear (polynomial) bases can drastically improve the performance. You will also explore how the complexity of a basis affects both the training error and the test error.

4.1 Adding a Bias Variable

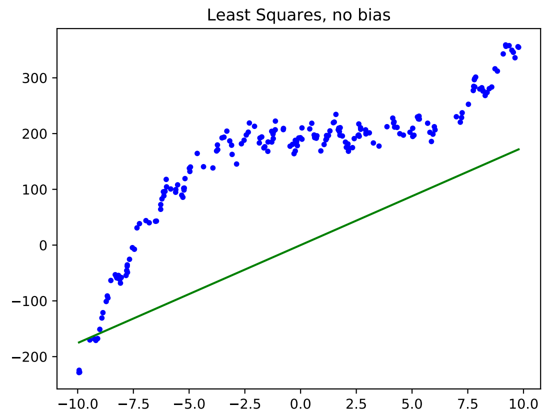
Rubric: {code:3}

If you run `python main.py -q 4`, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Report the training error.
4. Report the test error (on a dataset not used for training).

5. Draw a figure showing the training data and what the linear model looks like.

Unfortunately, this is an awful model of the data. The average squared training error on the data set is over 28000 (as is the test error), and the figure produced by the demo confirms that the predictions are usually nowhere near the training data:



The y-intercept of this data is clearly not zero (it looks like it's closer to 200), so we should expect to improve performance by adding a *bias* (a.k.a. intercept) variable, so that our model is

$$y_i = w^T x_i + w_0.$$

instead of

$$y_i = w^T x_i.$$

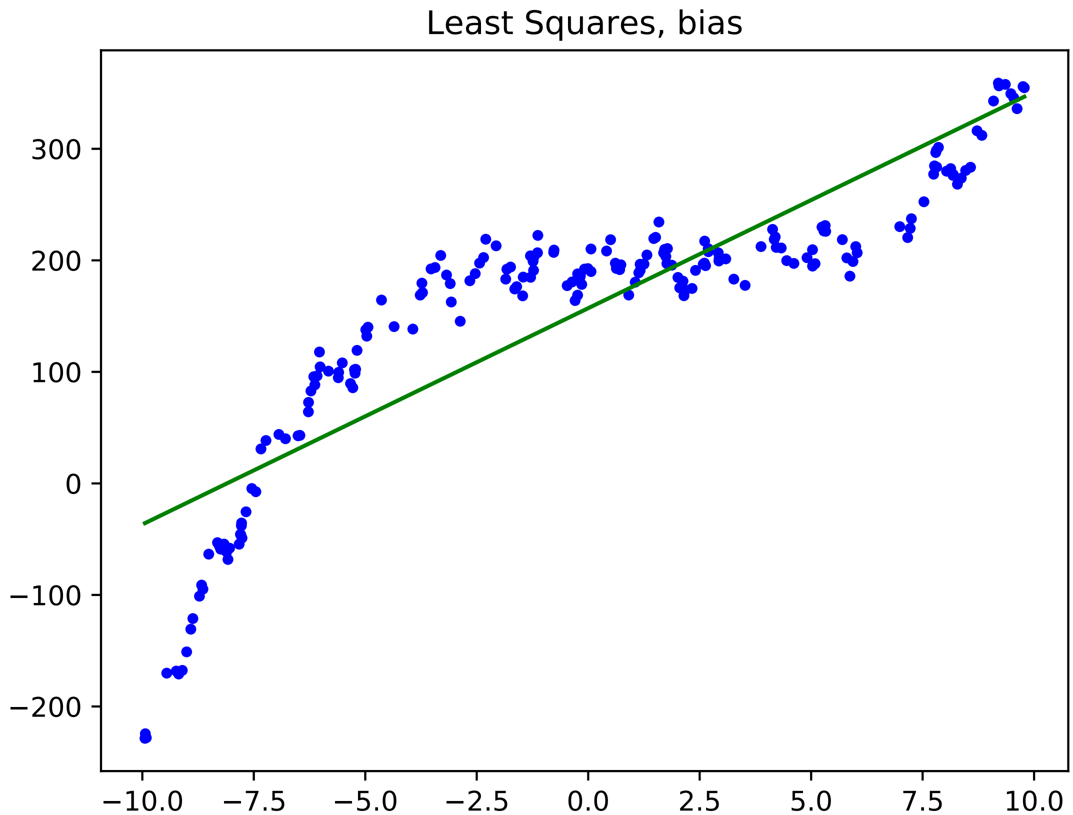
In file `linear_model.py`, complete the class, `LeastSquaresBias`, that has the same input/model/predict format as the `LeastSquares` class, but that adds a *bias* variable (also called an intercept) w_0 (also called β in lecture). Hand in your new class, the updated plot, and the updated training/test error.

Hint: recall that adding a bias w_0 is equivalent to adding a column of ones to the matrix X. Don't forget that you need to do the same transformation in the `predict` function.

```
# Least Squares with a bias added
class LeastSquaresBias:

    def fit(self,X,y):
        w_0 = np.ones(X.shape[0])
        X_ = np.c_[w_0,X]
        self.w = solve(X_.T@X_, X_.T@y)

    def predict(self, X):
        w_0 = np.ones(X.shape[0])
        X_ = np.c_[w_0,X]
        return X_@self.w
```



Model	Train error	Test error
LeastSquares	28 122.8	28 299.0
LeastSquaresBias	3551.3	3393.9

4.2 Polynomial Basis

Rubric: {code:4}

Adding a bias variable improves the prediction substantially, but the model is still problematic because the target seems to be a *non-linear* function of the input. Complete `LeastSquarePoly` class, that takes a data vector x (i.e., assuming we only have one feature) and the polynomial order p. The function should perform a least squares fit based on a matrix Z where each of its rows contains the values $(x_i)^j$ for $j = 0$ up to p. E.g., `LeastSquaresPoly.fit(x,y)` with $p = 3$ should form the matrix

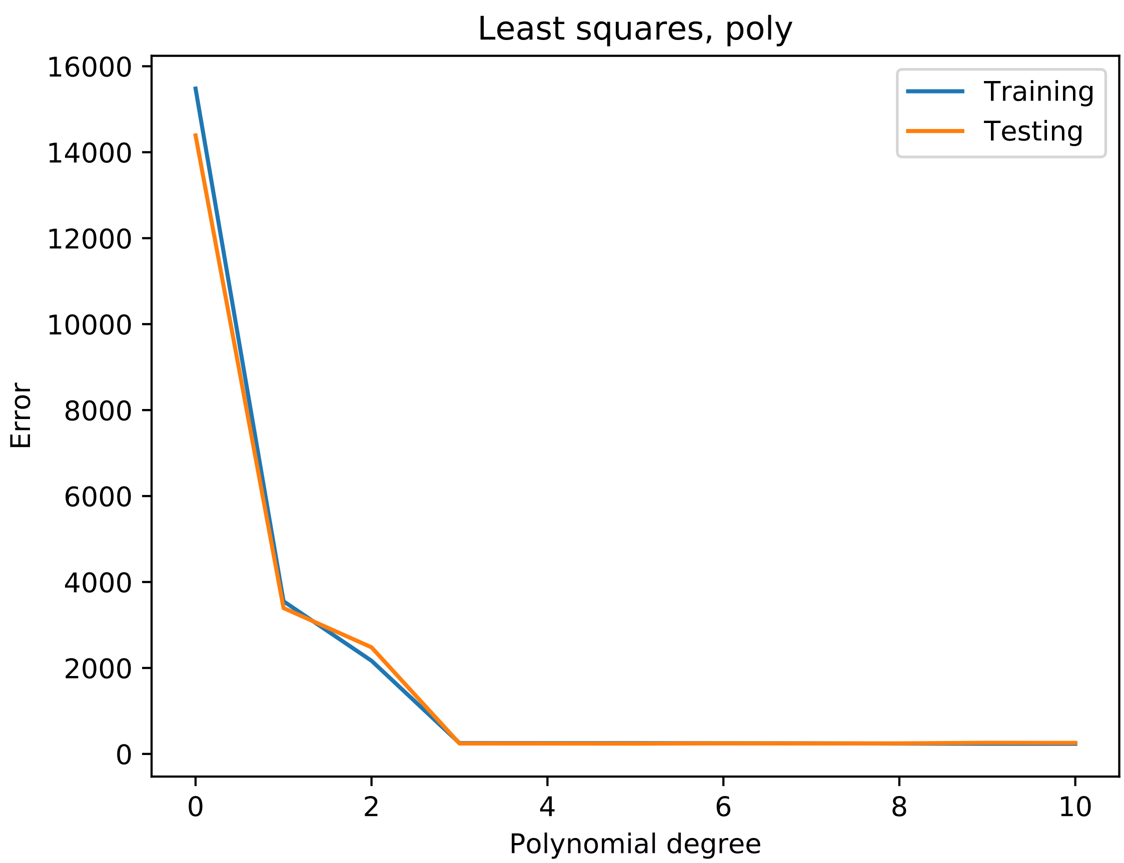
$$Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & (x_1)^3 \\ 1 & x_2 & (x_2)^2 & (x_2)^3 \\ \vdots & & & \\ 1 & x_n & (x_n)^2 & (x_N)^3 \end{bmatrix},$$

and fit a least squares model based on it. Hand in the new class, and report the training and test error for $p = 0$ through $p = 10$. Explain the effect of p on the training error and on the test error.

Note: you should write the code yourself; don't use a library like sklearn's `PolynomialFeatures`.

```
def predict(self, X):
    return self.leastSquares.predict(self.__polyBasis(X))

# A private helper function to transform any matrix X into
# the polynomial basis defined by this class at initialization
# Returns the matrix Z that is the polynomial basis of X.
def __polyBasis(self, X):
    z = np.ones((X.shape[0],1))
    for i in range(self.p):
        for j in range(X.shape[1]):
            z = np.c_[z,X[:,j]**(i+1)]
    return z
```



p	Train error	Test error
$p = 0$	15 480.5	14 390.763
$p = 1$	3551.34	3393.869
$p = 2$	2167.99	2480.725
$p = 3$	252.046	242.804
$p = 4$	251.461	242.126
$p = 5$	251.143	239.544
$p = 6$	248.582	246.005
$p = 7$	247.011	242.887
$p = 8$	241.306	245.966
$p = 9$	235.761	259.296
$p = 10$	235.074	256.299

As the value of p grows the error (training and testing) of the model decreases significantly, until $p = 3$ from there the training errors start decreasing slowly around 250 and on $p = 7$ the training error continue decreasing whereas the testing error start to increase. The model is just overfitting as p grows.

5 Very-Short Answer Questions

Rubric: {reasoning:7}

1. Suppose that a training example is global outlier, meaning it is really far from all other data points. How is the cluster assignment of this example by k-means? And how is it set by density-based clustering?
   k-means will cluster it to the "nearest" cluster it founds whereas density-based clustering will not assigned it to any cluster, because it may not be at the *eps* distance of any other training point.

2. Why do need random restarts for k-means but not for density-based clustering?
   Because k-means its severely affected by the initial position of the mean points, it may find different clusters if the two mean points are initialized pretty close. In the other hand density-based clustering do not depend on any position, but the hyperparameters *eps* and min_pts. If those hyperparameters do not change, the results of the density-based clustering will be the same on every run.

3. Can hierarchical clustering find non-convex clusters?
   It depends on the metrics used to create the clusters, then the hierarchical clusters found may or not be convex.

4. For model-based outlier detection, list an example method and problem with identifying outliers using this method.
   An example method will be using the z-score $z_i = \frac{x_i - \mu}{\sigma}$, to calculate the number of standard deviations away from the mean. The problem with this method is that $\mu\sigma$ are sensitive to outliers, so maybe an outlier would move so much the μ that we will not identify it correctly as an outlier.

5. For graphical-based outlier detection, list an example method and problem with identifying outliers using this method.
   An example method will be using an scatterplot to visualize the data set and decide what points are outliers. The problem of this method arises when the dimension of the feature grows. Because it is not easy to visualize data in more than $3d$, the method may have problems with datasets with high number of features.

6. For supervised outlier detection, list an example method and problem with identifying outliers using this method.
   We can use random forests to detect outliers, the problem with this is that we need to either add outliers to the training set or first identify some outliers using another detection method. Ahd then create the training set with the labels of outlier and not-outlier.

7. If we want to do linear regression with 1 feature, explain why it would or would not make sense to use gradient descent to compute the least squares solution.

By using just the normal equations the model is not robust enough to detect outliers so it may not be a great model to predict the data.

8. Why do we typically add a column of 1 values to X when we do linear regression? Should we do this if we're using decision trees?
   To add a *bias* or *intercept* to the model, with this we enable the model to not necessarily pass through the origin.

9. If a function is convex, what does that say about stationary points of the function? Does convexity imply that a stationary points exists?
   If a function is convex then every stationary point will be a global minima. And if the function is convex will always have stationary points.

10. Why do we need gradient descent for the robust regression problem, as opposed to just using the normal equations? Hint: it is NOT because of the non-differentiability. Recall that we used gradient descent even after smoothing away the non-differentiable part of the loss.
    Because it is faster to use rather than solving the normal equations.

11. What is the problem with having too small of a learning rate in gradient descent?
    It may take a lot of time to converge to the minima, because it will approximate to it at a slow pace.

12. What is the problem with having too large of a learning rate in gradient descent?
    It may oscillate in the function while approximating to the minima, which will also make longer the time of convergence.

13. What is the purpose of the log-sum-exp function and how is this related to gradient descent?
    To make an smooth approximation of the L_∞ norm, because minimization of the max function is much harder, we create an smooth approximation using $log()$ and $exp()$ which will be much similar to the max function and we can use gradient descent on it.

14. What type of non-linear transform might be suitable if we had a periodic function?
    A trigonometric transform will help with the periodic function.