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

Yihao Zhang, #82626169, u6m1b
Hafsa Binte Zahid, #11668150, i6r0b

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.

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`.
The name is Classic Accessories 73942 Veranda Grill Cover, X-Large, P.

[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.

The total stars are 14454.0

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.
The user A100WO06OQR8BQ has rated 161 items.

1.1.3 Histograms

Rubric: {code:2}

Make the following histograms:

1. The number of ratings per user

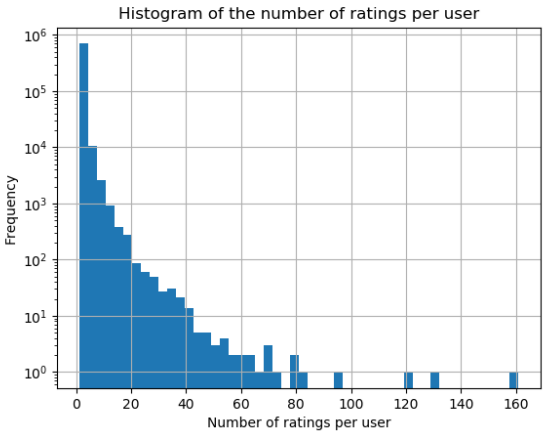


Figure 1: Histogram for number of ratings per user

2. The number of ratings per item

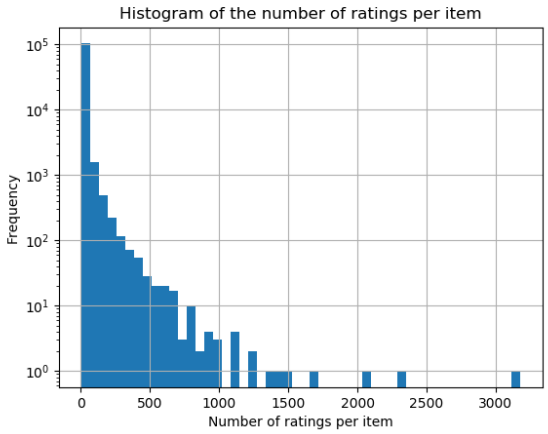


Figure 2: Histogram for number of ratings per item

3. The ratings themselves

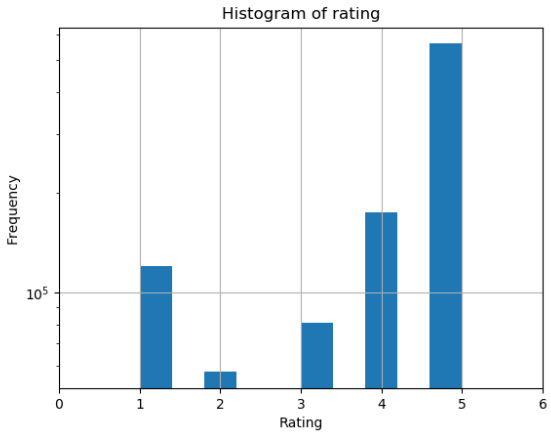


Figure 3: Histogram for the ratings

Code for Q1.1:

```
elif question == "1.1":
    filename = "ratings_Patio_Lawn_and_Garden.csv"
    with open(os.path.join("..", "data", filename), "rb") as f:
        ratings = pd.read_csv(f,names=("user","item","rating","timestamp"))
    X, user_mapper, item_mapper, user_inverse_mapper, \
    item_inverse_mapper, user_ind, item_ind = \
        utils.create_user_item_matrix(ratings)
    X_binary = X != 0

    # YOUR CODE HERE FOR Q1.1.
    stars = ratings.groupby('item').agg({'rating': 'sum'})
    print(stars[stars['rating'] == stars['rating'].max()])
```

```
# YOUR CODE HERE FOR Q1.1.2
reviews = ratings.groupby('user').agg({'item': 'count'})
print(reviews[reviews['item'] == reviews['item'].max()])

# YOUR CODE HERE FOR Q1.1.3
# The number of ratings per user
plt.figure()
reviews.hist(bins=50)
plt.yscale('log', nonposy='clip')
plt.xlabel('Number of ratings per user')
plt.ylabel('Frequency')
plt.title('Histogram of the number of ratings per user')
fname = os.path.join("..", "figs", "q1_1_3_1.png")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)

# The number of ratings per item
rating_per_item = ratings.groupby('item')['user'].nunique()
plt.figure()
rating_per_item.hist(bins=50)
plt.yscale('log', nonposy='clip')
plt.xlabel('Number of ratings per item')
plt.ylabel('Frequency')
plt.title('Histogram of the number of ratings per item')
fname = os.path.join("..", "figs", "q1_1_3_2.png")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)

# The rating themselves
plt.figure()
ratings.hist(column='rating')
plt.yscale('log', nonposy='clip')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.title('Histogram of rating')
fname = os.path.join("..", "figs", "q1_1_3_3.png")
plt.savefig(fname)
print("\nFigure saved as '%s'" % fname)
```

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: 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.

1. Euclidean distance (the `NearestNeighbors` default)
   For euclidean distance, top similar items are:
   https://www.amazon.com/dp/B00IJB5MCS
   https://www.amazon.com/dp/B00IJB4MLA
   https://www.amazon.com/dp/B00EXE4O42
   https://www.amazon.com/dp/B00743MZCM
   https://www.amazon.com/dp/B00HVXQY9A

2. Normalized Euclidean distance (you'll need to do the normalization)
   For normalize euclidean distance, top similar items are:
   https://www.amazon.com/dp/B00IJB5MCS
   https://www.amazon.com/dp/B00IJB8F3G
   https://www.amazon.com/dp/B00IJB4MLA
   https://www.amazon.com/dp/B00EF45AHU
   https://www.amazon.com/dp/B00EF3YF0Y

3. Cosine similarity (by setting `metric='cosine'`)
   For cosine similarity, top similar items are:
   https://www.amazon.com/dp/B00IJB5MCS
   https://www.amazon.com/dp/B00IJB8F3G
   https://www.amazon.com/dp/B00IJB4MLA
   https://www.amazon.com/dp/B00EF45AHU
   https://www.amazon.com/dp/B00EF3YF0Y

code for Q1.2:

```
elif question == "1.2":
    filename = "ratings_Patio_Lawn_and_Garden.csv"
    with open(os.path.join("..", "data", filename), "rb") as f:
        ratings = pd.read_csv(f,names=("user","item","rating","timestamp"))
    X, user_mapper, item_mapper, user_inverse_mapper, \
    item_inverse_mapper, user_ind, item_ind = \
        utils.create_user_item_matrix(ratings)
    X_binary = X != 0

    grill_brush = "B00CFM0P7Y"
    grill_brush_ind = item_mapper[grill_brush]
    grill_brush_vec = X[:,grill_brush_ind]
    print(url_amazon % grill_brush)

    # YOUR CODE HERE FOR Q1.2
    # Euclidean distance
```

```
print("For euclidean distance, top similar items are:")

model = NearestNeighbors()
model.fit(X.transpose())
ed_distances, ed_indices = model.kneighbors(grill_brush_vec.transpose(),
                                            n_neighbors=6)

for i in np.arange(1, len(ed_indices[0])):
    print(url_amazon % item_inverse_mapper[ed_indices[0, i]])

# Normalize euclidean distance
print("For normalize euclidean distance, top similar items are:")
n = len(set(ratings['user']))
d = len(set(ratings['item']))
sum = X.sum(axis=1)
X_norm = normalize(X, norm='l2', axis=0)

model.fit(X_norm.transpose())
grill_brush_vec_norm = normalize(grill_brush_vec,
                                 norm='l2',
                                 axis=0)
ned_distances, ned_indices = \
    model.kneighbors(grill_brush_vec_norm.transpose(),
                     n_neighbors=6)
for i in np.arange(1, len(ned_indices[0])):
    print(url_amazon % item_inverse_mapper[ned_indices[0, i]])

# Cosine similarity
print("For cosine similarity, top similar items are:")
model = NearestNeighbors(metric='cosine')
model.fit(X.transpose())
cos_distances, cos_indices = \
    model.kneighbors(grill_brush_vec.transpose(),
                     n_neighbors=6)
for i in np.arange(1, len(cos_indices[0])):
    print(url_amazon % item_inverse_mapper[cos_indices[0, i]])
```

Did normalized Euclidean distance and cosine similarity yields the same similar items, as expected?
It is expected because cosine similarity is equivalent to normalized KNN using euclidean distance.

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`.
For euclidean distance recommended items:
The item B00IJB5MCS has 55 reviews
The item B00IJB4MLA has 45 reviews
The item B00EXE4O42 has 1 reviews
The item B00743MZCM has 1 reviews

The item B00HVXQY9A has 1 reviews
For cosine similarity recommended items:
The item B00IJB5MCS has 55 reviews
The item B00IJB8F3G has 91 reviews
The item B00IJB4MLA has 45 reviews
The item B00EF45AHU has 66 reviews
The item B00EF3YF0Y has 110 reviews

In class, it is mentioned that normalization means it prefers the popular items. It also matches our results. For euclidean distance, item 3, 4 and 5 only have 1 review. However, all the recommended items by cosine similarity have larger numbers of reviews which improve the recommend accuracy.
With un-normalized euclidean distances, the stars of an item is given more value than the number of users who rated it. So, items with only one rating but 5 stars are returned as near neighbours. Using cosine differences allows us to take both the stars and the number of reviews into account. This gives a better picture of how popular an item is; an item with a slightly lower rating but rated by a lot of people is given better rankings than one with higher rating but much fewer users.

Code for Q1.3:

```
# YOUR CODE HERE FOR Q1.3
        # For euclidean distance
        print("For euclidean distance recommended items:")
        for i in np.arange(1, len(ed_distances[0])):
            item_code = item_inverse_mapper[ed_indices[0, i]]
            print("The item {} has {} reviews"
                  .format(item_code,
                          ratings['item'].value_counts().loc[item_code]))

        # For cosine similarity
        print("For cosine similarity recommended items:")
        for i in np.arange(1, len(cos_distances[0])):
            item_code = item_inverse_mapper[cos_indices[0, i]]
            print("The item {} has {} reviews"
                  .format(item_code,
                          ratings['item'].value_counts().loc[item_code]))
```

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. $\max_{i\in\{1,2,\dots,n\}} |w^T x_i - y_i|$.
   $\|Xw - y\|_\infty$
2. $\sum_{i=1}^n v_i(w^T x_i - y_i)^2 + \frac{\lambda}{2}\sum_{j=1}^d w_j^2$.
   $(Xw - y)^T V(Xw - y) + \frac{\lambda}{2} w^T w$
3. $\left(\sum_{i=1}^n |w^T x_i - y_i|\right)^2 + \frac{1}{2}\sum_{j=1}^d \lambda_j |w_j|$.
   $|Xw - y|_1^2 + \frac{1}{2}|\Lambda w|_1$

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).

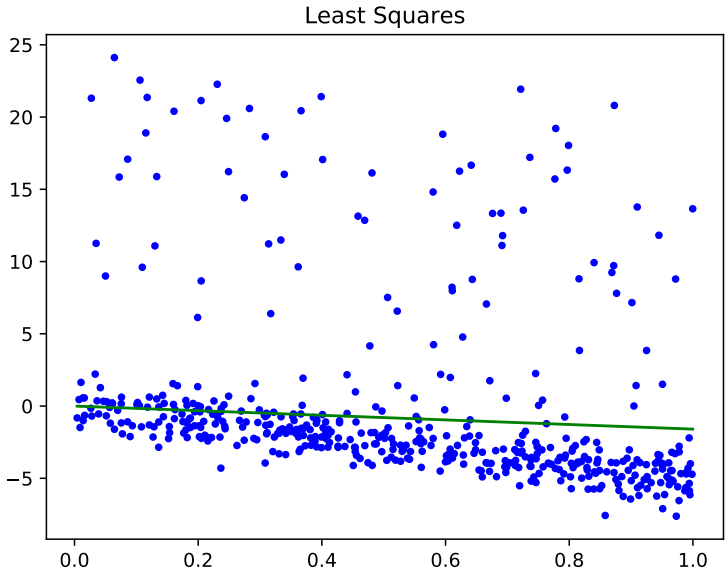
1. $f(w) = \frac{1}{2}\|w - v\|^2$ (projection of v onto real space).
   $f(w) = \frac{1}{2}||w||^2 + \frac{1}{2}||v||^2 - w^T v$
   $\nabla f(w) = w - v = 0$
   Thus, $w = v$
2. $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^T \Lambda w$ (least squares with weighted regularization).
   $f(w) = ||Xw||^2 + ||y||^2 - (Xw)^T y + \frac{1}{2}w^T \Lambda w$
   $\nabla f(w) = X^T Xw - X^T y + \Lambda w$
   Thus, solve w for $(X^T X + \Lambda)w = X^T y$
3. $f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2 + \frac{\lambda}{2}\|w - w^0\|^2$ (weighted least squares shrunk towards non-zero w^0).
   $f(w) = \frac{1}{2}(Xw - y)^T V(Xw - y) + \frac{\lambda}{2}(||w||^2 + ||w^0||^2 - 2w^T w^0)$
   $f(w) = \frac{1}{2}(w^T X^T VXw) - \frac{1}{2}(w^T X^T Vy) - \frac{1}{2}(y^T VXw) + \frac{1}{2}(y^T Vy) + \frac{\lambda}{2}(||w||^2 + ||w^0||^2 - 2w^T w^0)$
   $\nabla f(w) = X^T VXw - X^T Vy + \lambda w - \lambda w^0$
   Thus, solve w for $(X^T VX + \lambda I)w = X^T Vy + \lambda w^0$

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, $X^T Xw + \lambda w$ can be re-written as $(X^T X + \lambda I)w$.

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.

$$
\begin{aligned}
f(w) &= \frac{1}{2}(w^T X^T - y^T)(VXw - Vy) \\
&= \frac{1}{2}w^T(X^T V X)w - \frac{1}{2}w^T X^T V y - \frac{1}{2}y^T V X w + \frac{1}{2}V^T V y \\
\nabla f(w) &= X^T V X w - \frac{1}{2}X^T V y - \frac{1}{2}y^T V X \\
&= X^T V X w - X^T V y
\end{aligned}
$$

solve for $X^T V X w - X^T V y = 0$

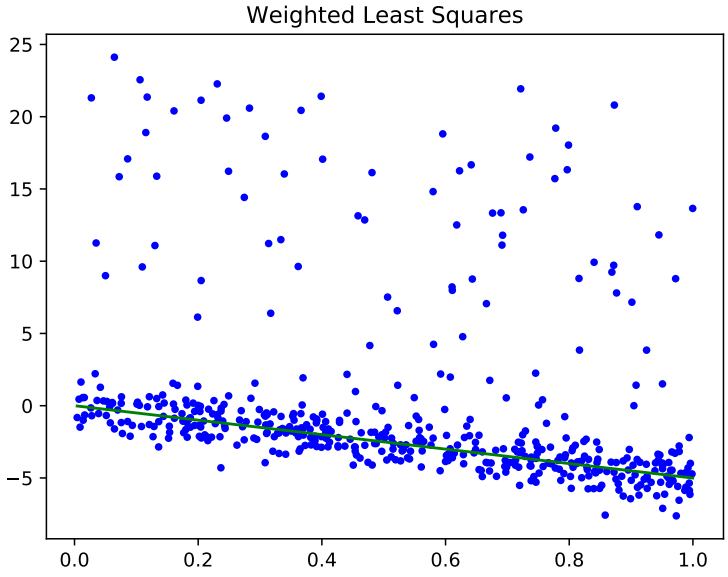


Figure 4: Weighted Least Squares Fitting

Code for Q3.1:

```
elif question == "3.1":
    data = load_dataset("outliersData.pkl")
    X = data['X']
    y = data['y']

    len = len(X)
    v = np.full(shape=len, fill_value=1)
    v[400:500] = 0.1
    V = np.diag(v)

    model = linear_model.WeightedLeastSquares()
    model.fit(X, y, V)
    utils.test_and_plot(model,X,y,
                        title="Weighted Least Squares",
```

```
                                filename="weight_least_squares_outliers.pdf")
```

WeightedLeastSquares class:

```
class WeightedLeastSquares(LeastSquares): # inherits the predict() function from LeastSquares
    def fit(self,X,y,V):
        ''' YOUR CODE HERE '''
        self.w = solve(X.T@V@X, X.T@V@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.

Since $w^T x_i$ is a value and y_i is a value, and w is a $d * 1$ vector, x_i must be a vector and X must be a matrix.

$$\frac{df}{dr} = \sum_{i=1}^{n} \frac{\exp(r) - \exp(-r)}{\exp(r) + \exp(-r)}$$

$$\text{With } r = w^T x_i - y_i$$

$$\frac{dr}{dw_j} = x_{ij}$$

By chain rule:

$$\frac{df}{dw_j} = \frac{df}{dr} * \frac{dr}{dw_j}$$

$$= \sum_{i=1}^{n} \left(\frac{exp(r) - exp(-r)}{exp(r) + exp(-r)} * x_{ij}\right)$$

$$= \sum_{i=1}^{n} \left(\frac{\exp(w^T x_i - y_i) - \exp(-w^T x_i + y_i)}{\exp(w^T x_i - y_i) + \exp(-w^T x_i + y_i)} * x_{ij}\right)$$

Using the above, we get $\frac{df}{dw_j}$ for each row $j, j \in [1, d]$ in the vector w.

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

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.

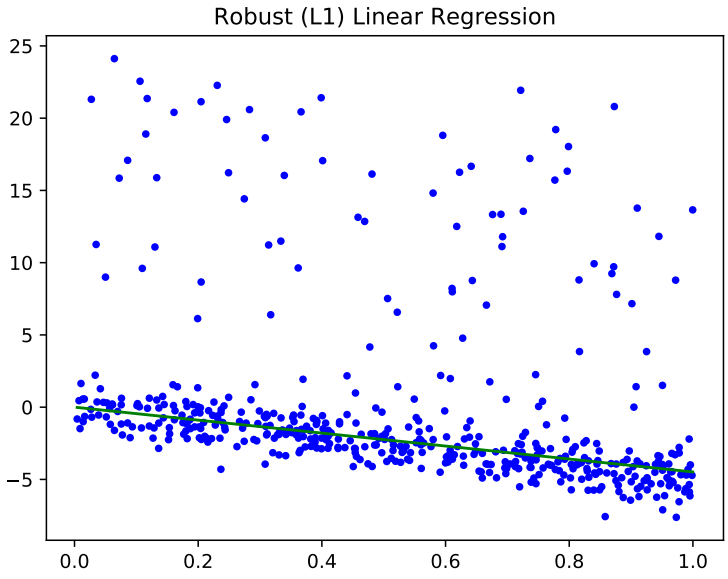


Figure 5: Robust Regression Fitting

```
def funObj(self,w,X,y):
    # Calculate the function value
    f = np.zeros(len(X[0]))
    for j in range(len(X[0])):
        _sum = 0
        for i in range(len(X)):
```

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

```
            tmp = w.T * X[i].T - y[i]
            _sum += np.sum(np.log(np.exp(tmp) + np.exp(-tmp)))
        f[j] = _sum

    # Calculate the gradient value
    g = np.zeros(len(X[0]))
    for j in range(len(X[0])):
        _sum = 0
        for i in range(len(X)):
            tmp =w.T * X[i].T - y[i]
            _sum += (np.exp(tmp) - np.exp(-tmp))/(np.exp(tmp) + np.exp(-tmp)) * X[i, j]
        g[j] = _sum
    return f, g
```

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.

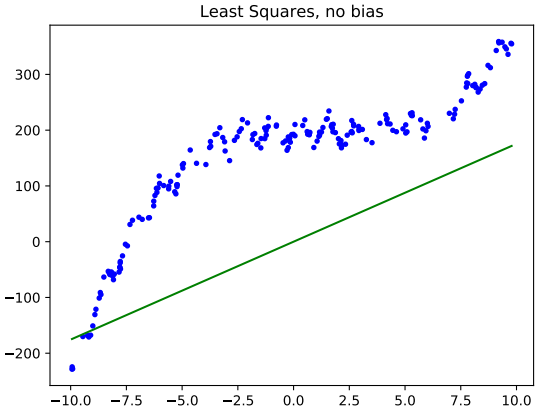
4.1 Adding a Bias Variable

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.
Training error = 3551.3
Test error = 3393.9

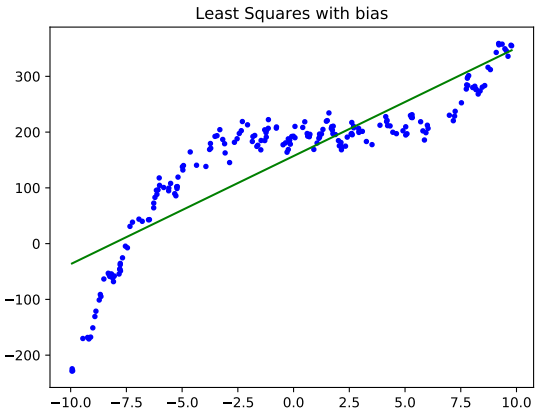


Figure 6: Least Square Fitting With Bias Added

LeastSquaresBias class:

```
class LeastSquaresBias:

    def fit(self,X,y):
        w0 = np.ones(shape=(len(X), 1))
        self.Z = np.concatenate((X, w0), axis=1)
        self.w = solve(self.Z.T@self.Z, self.Z.T@y)

    def predict(self, X):
        X = np.concatenate((X, np.ones(shape=(len(X), 1))), axis=1)
        return X@self.w
```

main.py for Q4.1:

```
    elif question == "4.1":
        data = load_dataset("basisData.pkl")
        X = data['X']
        y = data['y']
        Xtest = data['Xtest']
        ytest = data['ytest']

        # YOUR CODE HERE
        model = linear_model.LeastSquaresBias()
        model.fit(X, y)

        utils.test_and_plot(model,X,y,Xtest,ytest,
                            title="Least Squares with bias",
                            filename="least_squares_bias.pdf")
```

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`.
p=0, Training error = 28122.8 Test error = 28299.0
p=1 Training error = 3551.3 Test error = 3393.9
p=2 Training error = 2168.0 Test error = 2480.7
p=3 Training error = 252.0 Test error = 242.8
p=4 Training error = 251.5 Test error = 242.1
p=5 Training error = 251.1 Test error = 239.5
p=6 Training error = 248.6 Test error = 246.0

p=7 Training error = 247.0 Test error = 242.9
p=8 Training error = 241.3 Test error = 246.0
p=9 Training error = 235.8 Test error = 259.3
p=10 Training error = 235.1 Test error = 256.3

With higher value of p, we are fitting higher polynormial. The training error will keep reducing when p gets larger. When p = number of examples, the training error will be 0 and the fitting will go through every point exactly. However, with large p, the model starts to overfit. And that is the reason why test error starts to grow near the end.

LeastSquaresPoly class:

```
class LeastSquaresPoly:
    def __init__(self, p):
        self.leastSquares = LeastSquares()
        self.p = p

    def fit(self,X,y):
        self.__polyBasis(X)
        self.w = solve(self.Z.T@self.Z, self.Z.T@y)

    def predict(self, X):
        self.__polyBasis(X)
        return self.Z@self.w

    # 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):
        if self.p == 0:
            self.Z = X
            return

        new_cols = np.empty(shape=(len(X), self.p))
        w0 = np.ones(len(X))
        new_cols[:, 0] = w0

        for i in range(2, self.p + 1):
            new_col = np.power(X[:, 0], i)
            new_cols[:, i - 1] = new_col
        self.Z = np.concatenate((X, new_cols), axis=1)
```

main.py for Q4.2:

```
    elif question == "4.2":
        data = load_dataset("basisData.pkl")
        X = data['X']
        y = data['y']
        Xtest = data['Xtest']
        ytest = data['ytest']

        for p in range(11):
            print("p=%d" % p)

            # YOUR CODE HERE
            model = linear_model.LeastSquaresPoly(p)
            model.fit(X, y)
```

```
file_name = "least_square_poly_" + str(p)
title = "Least Squares with polynomial of " + str(p)
utils.test_and_plot(model,X,y,Xtest,ytest,
                    title=title,filename=file_name)
```

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?
   Using k-means, the number of clusters is defined by k. With a global outlier, it will affect the location of the cluster since it changes the mean value a lot. Thus, it will change the cluster location further away from the original data points belong to such cluster.
   With density-based clustering, this global outlier will not be in any cluster and be treated as an outlier.

2. Why do need random restarts for k-means but not for density-based clustering?
   For k-means, it has the possibility to converge to sub-optimal solution. Random starts can reduce the possibility leading to sub-optimal solution. Depending on what the initial points for k means are, it converges differently.
   Density-based clustering is not sensitive to initialization except for boundary points. The convergences always starts and ends the same.

3. Can hierarchical clustering find non-convex clusters?
   Hierarchical clustering can find non-convex clusters. While merging two clusters together, the resulting cluster is not guaranteed to be convex.

4. For model-based outlier detection, list an example method and problem with identifying outliers using this method.
   Example: Given mean and variance of the data, z-score can be computed and used as outlier detector. For example, if $|z| >$ some threshold, then treat it as an outlier.
   Problem: Mean and variance are sensitive to outliers. if the outlier's value is large and changes mean and variance a lot, z-score won't be very accurate.

5. For graphical-based outlier detection, list an example method and problem with identifying outliers using this method.
   Example: Given boxplot or scatter plot, human looks at the data and decide if the data is an outlier or not.
   Problem: Will be laborious in high-dimensions.

6. For supervised outlier detection, list an example method and problem with identifying outliers using this method.
   Example: Given data with labels of outlier/not outlier, train the model and use the model to detect outliers.
   Problem: We need to know what the outlier looks like and we need labelled data. We may not able to detect "new" types of outliers.

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.
   It wouldn't make sense to use gradient descent in this case in terms of time complexity. Gradient descent takes $O(ndt)$ whereas normal equations takes $O(nd^2 + d^3)$. With $d = 1$, normal equations will be faster to compute the minimizer w.

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?
   In linear regression, adding a column of 1 makes the intercept to be non-zero. Without it, for all $x_i = 0$, we will predict corresponding $y_i = 0$. We shouldn't add it for decision trees because adding a feature with the same value for all columns will not affect the decision outcome

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, it means that all stationary points are minimum points, and all local minimum points are global minimum points. However, convexity does not imply that a stationary point exists at all - this is dependent on the range of values over which the function is defined. A strictly decreasing convex function, for example, will not have a stationary point.

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.
    For robust regression, we must take the L1-norm, and this does not have a representation in normal equations. So gradient decent is our only option for finding the minimum.

11. What is the problem with having too small of a learning rate in gradient descent?
    Training will take longer time for the model to converge.

12. What is the problem with having too large of a learning rate in gradient descent?
    The model may not be able to converge.

13. What is the purpose of the log-sum-exp function and how is this related to gradient descent?
    The log-sum-exp is a smooth approximation of infinity-norm objective function (max). When infinity-norm is used as objective, it is non-differential at 0 where the gradient descent will have trouble. The purpose of log-sum-exp function is to smooth the objection function mainly around 0 so the gradient descent can be applied.

14. What type of non-linear transform might be suitable if we had a periodic function?
    We could try to transform to a different coordinate system, for example if we had features x and y and they were periodic, we could transform to polar coordinates such that x=r*cos(theta) and y=r*sin(theta), to create a new matrix Z that is linear in r and theta. This can be done in spherical and possibly higher dimensions as well

Appendices

A main.py

```
# basics
import argparse
import os
import pickle
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from scipy.sparse import csr_matrix as sparse_matrix

# sklearn imports
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import normalize

# our code
import linear_model
import utils

url_amazon = "https://www.amazon.com/dp/%s"

def load_dataset(filename):
    with open(os.path.join('..','data',filename), 'rb') as f:
        return pickle.load(f)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-q','--question', required=True)
    io_args = parser.parse_args()
    question = io_args.question

    if question == "1":

        filename = "ratings_Patio_Lawn_and_Garden.csv"
        with open(os.path.join("..", "data", filename), "rb") as f:
            ratings = pd.read_csv(f,names=("user","item","rating","timestamp"))

        print("Number of ratings:", len(ratings))
        print("The average rating:", np.mean(ratings["rating"]))

        n = len(set(ratings["user"]))
        d = len(set(ratings["item"]))
        print("Number of users:", n)
        print("Number of items:", d)
        print("Fraction nonzero:", len(ratings)/(n*d))

        X, user_mapper, item_mapper, user_inverse_mapper, \
        item_inverse_mapper, user_ind, item_ind = \
            utils.create_user_item_matrix(ratings)
```

```
    print(type(X))
    print("Dimensions of X:", X.shape)

elif question == "1.1":
    filename = "ratings_Patio_Lawn_and_Garden.csv"
    with open(os.path.join("..", "data", filename), "rb") as f:
        ratings = pd.read_csv(f,names=("user","item","rating","timestamp"))
    X, user_mapper, item_mapper, user_inverse_mapper, \
    item_inverse_mapper, user_ind, item_ind = \
        utils.create_user_item_matrix(ratings)
    X_binary = X != 0

    # YOUR CODE HERE FOR Q1.1.
    stars = ratings.groupby('item').agg({'rating': 'sum'})
    print(stars[stars['rating'] == stars['rating'].max()])

    # YOUR CODE HERE FOR Q1.1.2
    reviews = ratings.groupby('user').agg({'item': 'count'})
    print(reviews[reviews['item'] == reviews['item'].max()])

    # YOUR CODE HERE FOR Q1.1.3
    # The number of ratings per user
    plt.figure()
    reviews.hist(bins=50)
    plt.yscale('log', nonposy='clip')
    plt.xlabel('Number of ratings per user')
    plt.ylabel('Frequency')
    plt.title('Histogram of the number of ratings per user')
    fname = os.path.join("..", "figs", "q1_1_3_1.png")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

    # The number of ratings per item
    rating_per_item = ratings.groupby('item')['user'].nunique()
    plt.figure()
    rating_per_item.hist(bins=50)
    plt.yscale('log', nonposy='clip')
    plt.xlabel('Number of ratings per item')
    plt.ylabel('Frequency')
    plt.title('Histogram of the number of ratings per item')
    fname = os.path.join("..", "figs", "q1_1_3_2.png")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

    # The rating themselves
    plt.figure()
    ratings.hist(column='rating')
    plt.yscale('log', nonposy='clip')
    plt.xlabel('Rating')
    plt.ylabel('Frequency')
    plt.title('Histogram of rating')
    fname = os.path.join("..", "figs", "q1_1_3_3.png")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)
```

```
elif question == "1.2":
    filename = "ratings_Patio_Lawn_and_Garden.csv"
    with open(os.path.join("..", "data", filename), "rb") as f:
        ratings = pd.read_csv(f,names=("user","item","rating","timestamp"))
    X, user_mapper, item_mapper, user_inverse_mapper, \
    item_inverse_mapper, user_ind, item_ind = \
        utils.create_user_item_matrix(ratings)
    X_binary = X != 0

    grill_brush = "B00CFM0P7Y"
    grill_brush_ind = item_mapper[grill_brush]
    grill_brush_vec = X[:,grill_brush_ind]
    print(url_amazon % grill_brush)

    # YOUR CODE HERE FOR Q1.2
    # Euclidean distance
    print("For euclidean distance, top similar items are:")

    model = NearestNeighbors()
    model.fit(X.transpose())
    ed_distances, ed_indices = model.kneighbors(grill_brush_vec.transpose(),
                                                n_neighbors=6)

    for i in np.arange(1, len(ed_indices[0])):
        print(url_amazon % item_inverse_mapper[ed_indices[0, i]])

    # Normalize euclidean distance
    print("For normalize euclidean distance, top similar items are:")
    n = len(set(ratings['user']))
    d = len(set(ratings['item']))
    sum = X.sum(axis=1)
    X_norm = normalize(X, norm='l2', axis=0)

    model.fit(X_norm.transpose())
    grill_brush_vec_norm = normalize(grill_brush_vec,
                                     norm='l2',
                                     axis=0)
    ned_distances, ned_indices = \
        model.kneighbors(grill_brush_vec_norm.transpose(),
                         n_neighbors=6)
    for i in np.arange(1, len(ned_indices[0])):
        print(url_amazon % item_inverse_mapper[ned_indices[0, i]])

    # Cosine similarity
    print("For cosine similarity, top similar items are:")
    model = NearestNeighbors(metric='cosine')
    model.fit(X.transpose())
    cos_distances, cos_indices = \
        model.kneighbors(grill_brush_vec.transpose(),
                         n_neighbors=6)
    for i in np.arange(1, len(cos_indices[0])):
        print(url_amazon % item_inverse_mapper[cos_indices[0, i]])

    # YOUR CODE HERE FOR Q1.3
    # For euclidean distance
    print("For euclidean distance recommended items:")
```

```
    for i in np.arange(1, len(ed_distances[0])):
        item_code = item_inverse_mapper[ed_indices[0, i]]
        print("The item {} has {} reviews"
              .format(item_code,
                      ratings['item'].value_counts().loc[item_code]))

    # For cosine similarity
    print("For cosine similarity recommended items:")
    for i in np.arange(1, len(cos_distances[0])):
        item_code = item_inverse_mapper[cos_indices[0, i]]
        print("The item {} has {} reviews"
              .format(item_code,
                      ratings['item'].value_counts().loc[item_code]))


elif question == "3":
    data = load_dataset("outliersData.pkl")
    X = data['X']
    y = data['y']

    # Fit least-squares estimator
    model = linear_model.LeastSquares()
    model.fit(X,y)
    print(model.w)

    utils.test_and_plot(model,X,y,title="Least Squares",
                        filename="least_squares_outliers.pdf")

elif question == "3.1":
    data = load_dataset("outliersData.pkl")
    X = data['X']
    y = data['y']

    len = len(X)
    v = np.full(shape=len, fill_value=1)
    v[400:500] = 0.1
    V = np.diag(v)

    model = linear_model.WeightedLeastSquares()
    model.fit(X, y, V)
    utils.test_and_plot(model,X,y,
                        title="Weighted Least Squares",
                        filename="weight_least_squares_outliers.pdf")

elif question == "3.3":
    # loads the data in the form of dictionary
    data = load_dataset("outliersData.pkl")
    X = data['X']
    y = data['y']

    # Fit least-squares estimator
    model = linear_model.LinearModelGradient()
    model.fit(X,y)
    print(model.w)

    utils.test_and_plot(model,X,y,title="Robust (L1) Linear Regression",
```

```
                        filename="least_squares_robust.pdf")

elif question == "4":
    data = load_dataset("basisData.pkl")
    X = data['X']
    y = data['y']
    Xtest = data['Xtest']
    ytest = data['ytest']

    # Fit least-squares model
    model = linear_model.LeastSquares()
    model.fit(X,y)

    utils.test_and_plot(model,X,y,Xtest,ytest,
                        title="Least Squares, no bias",
                        filename="least_squares_no_bias.pdf")

elif question == "4.1":
    data = load_dataset("basisData.pkl")
    X = data['X']
    y = data['y']
    Xtest = data['Xtest']
    ytest = data['ytest']

    # YOUR CODE HERE
    model = linear_model.LeastSquaresBias()
    model.fit(X, y)

    utils.test_and_plot(model,X,y,Xtest,ytest,
                        title="Least Squares with bias",
                        filename="least_squares_bias.pdf")

elif question == "4.2":
    data = load_dataset("basisData.pkl")
    X = data['X']
    y = data['y']
    Xtest = data['Xtest']
    ytest = data['ytest']

    for p in range(11):
        print("p=%d" % p)

        # YOUR CODE HERE
        model = linear_model.LeastSquaresPoly(p)
        model.fit(X, y)
        file_name = "least_square_poly_" + str(p)
        title = "Least Squares with polynomial of " + str(p)
        utils.test_and_plot(model,X,y,Xtest,ytest,
                            title=title,filename=file_name)


else:
    print("Unknown question: %s" % question)
```

B linear_model.py

```
import numpy as np
from numpy.linalg import solve
from findMin import findMin
from scipy.optimize import approx_fprime
import utils


# Ordinary Least Squares
class LeastSquares:
    def fit(self,X,y):
        self.w = solve(X.T@X, X.T@y)

    def predict(self, X):
        return X@self.w


# Least squares where each sample point X has a weight associated with it.
class WeightedLeastSquares(LeastSquares): # inherits the predict() function from LeastSquares
    def fit(self,X,y,V):
        ''' YOUR CODE HERE '''
        self.w = solve(X.T@V@X, X.T@V@y)


class LinearModelGradient(LeastSquares):
    def fit(self,X,y):
        n, d = X.shape

        # Initial guess
        self.w = np.zeros((d, 1))

        # check the gradient
        estimated_gradient = approx_fprime(self.w,
                                           lambda w: self.funObj(w,X,y)[0],
                                           epsilon=1e-6)
        implemented_gradient = self.funObj(self.w,X,y)[1]
        if np.max(np.abs(estimated_gradient - implemented_gradient) > 1e-4):
            print('User and numerical derivatives differ: %s vs. %s' %
                  (estimated_gradient, implemented_gradient));
        else:
            print('User and numerical derivatives agree.')

        self.w, f = findMin(self.funObj, self.w, 100, X, y)

    def funObj(self,w,X,y):
        # Calculate the function value
        f = np.zeros(len(X[0]))
        for j in range(len(X[0])):
            _sum = 0
            for i in range(len(X)):
                tmp = w.T * X[i].T - y[i]
                _sum += np.sum(np.log(np.exp(tmp) + np.exp(-tmp)))
            f[j] = _sum

        # Calculate the gradient value
```

```
        g = np.zeros(len(X[0]))
        for j in range(len(X[0])):
            _sum = 0
            for i in range(len(X)):
                tmp =w.T * X[i].T - y[i]
                _sum += (np.exp(tmp)
                         - np.exp(-tmp))/(np.exp(tmp)
                                          + np.exp(-tmp)) * X[i, j]
            g[j] = _sum
        return f, g


# Least Squares with a bias added
class LeastSquaresBias:

    def fit(self,X,y):
        w0 = np.ones(shape=(len(X), 1))
        self.Z = np.concatenate((X, w0), axis=1)
        self.w = solve(self.Z.T@self.Z, self.Z.T@y)

    def predict(self, X):
        X = np.concatenate((X, np.ones(shape=(len(X), 1))), axis=1)
        return X@self.w


# Least Squares with polynomial basis
class LeastSquaresPoly:
    def __init__(self, p):
        self.leastSquares = LeastSquares()
        self.p = p

    def fit(self,X,y):
        self.__polyBasis(X)
        self.w = solve(self.Z.T@self.Z, self.Z.T@y)

    def predict(self, X):
        self.__polyBasis(X)
        return self.Z@self.w

    # 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):
        if self.p == 0:
            self.Z = X
            return

        new_cols = np.empty(shape=(len(X), self.p))
        w0 = np.ones(len(X))
        new_cols[:, 0] = w0

        for i in range(2, self.p + 1):
            new_col = np.power(X[:, 0], i)
            new_cols[:, i - 1] = new_col
        self.Z = np.concatenate((X, new_cols), axis=1)
```