Machine Learning

Regressors Homework

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Purpose and notes

- In this homework, we will get hands on concepts such as polynomial regression, cross validation and grid search.
- In this project, use **sklean library** as much as you can
  - Their linear regression, ridge, lasso model, etc
- By default, we will use the minmax scaler
  - But your code should allow you experience with:
    - 1) Standardizing the data 2) no normalization
  - Be careful from data leaks
- **Fix** random state to 17 in sklearn
- **Fix** numpy with: np.random.seed(17)

Dataset

- **File**: data2_200x30.csv
- It consists of 200 rows and 31 columns
  - The first 30 columns are the **input** features (anonymous features)
  - The **last** column is the **target** response
  - **No feature engineering** should be done (outliers, duplicates, missing values, etc)
    - You will always scale the data using **MinMaxScaler**
- The data represents your trainval subset
  - Sometimes, you will use all together (e.g. for cross validation)
  - Manytimes: **use the first 100 examples** for **training** and remaining of data for **validation**
  - Data is already **shuffled** for you

Tasks

- Use argparse to provide the following functionalities, as explained in the next slides

```
parser.add_argument('--dataset', type=str, default='data2_200x30.csv')
parser.add_argument('--preprocessing', type=int, default=1,
                    help='0 for no processing, 1 for min/max scaling and 2 for standrizing')

parser.add_argument('--choice', type=int, default=6,
                    help='1 for simple linear regression, '
                         '2 for polynomial on all features with degrees [1, 2, 3, 4] with cross features'
                         '3 for polynomial on all features with degrees [1, 2, 3, 4] with monomial features'
                         '4 individual features test'
                         '5 for find best lambda with grid search'
                         '6 Lasso selection'
                    )
parser.add_argument('--extra_args', type=str, help='extra values passed', default='0,3,6')

args = parser.parse_args()
```

Task #1: KISS

- KISS, an acronym for "**Keep it simple, stupid**!", is a principle that encourages simplicity as possible
- Always start with making **simple** choices
- As a regression task, just try to use **all input features** with a **irregularized** linear regression model to regress the output
  - model = LinearRegression(fit_intercept = **True**)
- We call that a **baseline**
  - Use **RMSE** for reporting the error: **sqrt**(mean squared error)
  - Then in next the complex modeling, we see if we can beat the baseline
- Data:
  - Use the train/val given **split** to report validation performance for **minmax scaled data**
    - The first 100 examples for the train and the last 100 for validation

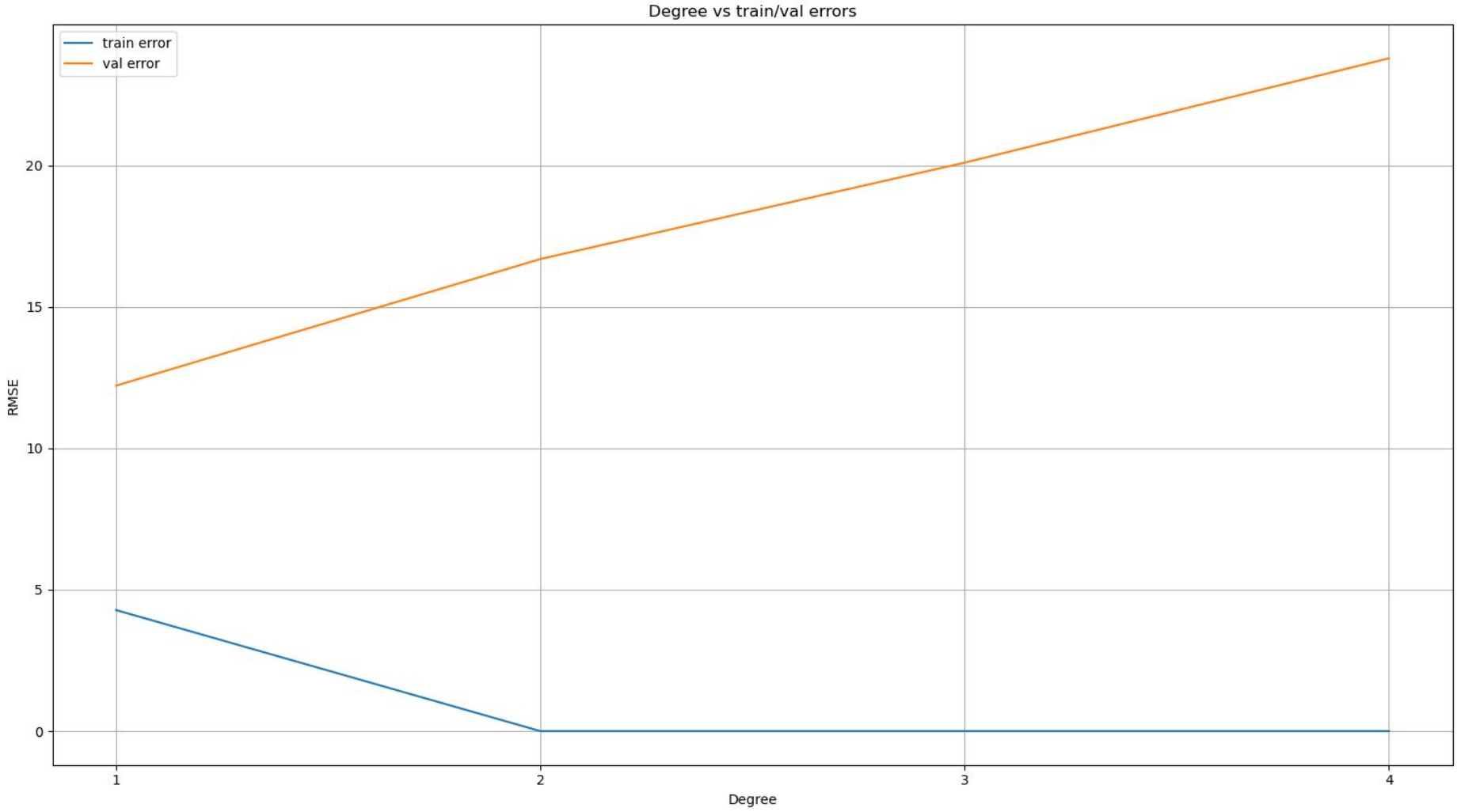
Task #1: KISS

- You should get the following values:
- intercept: [-19.75512349] - abs avg weight: 49.7237782166579 76
- **Error of train**: 4.28124973015366
- **Error of val**: 12.216701916378916

Task #2: Polynomial for all features v1

- Try **irregularized polynomial** regression
  - Polynomial degrees [1, 2, 3, 4]
  - Use sklearn PolynomialFeatures to compute the new features
  - Data: Same as Task #1 (100, 100)
- Evaluate the model on both train and val datasets using **RMSE**
- Show a graph with
  - X-axis: degrees [1, 2, 3, 4]
  - Y:axis: comparison of the train error
    vs the validation error

Polynomial of degree 3
- intercept: [28.09002879]
- Error of train: 1.539e-13
- Error of val: 20.10304


Degree vs train/val errors
train error
val error
RMSE
Degree
0
5
10
15
20
1
2
3
4

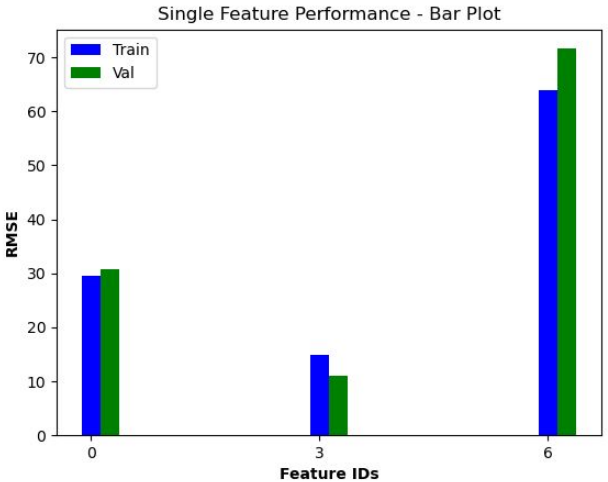
Task #3: Polynomial for all features v2

- You will notice, with a polynomial degree = 3, a 30 feature vector is converted to 5455 features due to using cross features
- We would like to generate non-cross features (e.g. monomials)
  - For each single feature x with degree 3, generate: x, x^2 and x^3
    - This means, for 30 features, we generate 90 features only
  - Implement a function def monomials_poly_features(X, degree)
    - It generates monomial features for X
- Repeat the previous task with non-cross features (numbers / graph)
- What are your thoughts?!

Polynomial of degree 3
- intercept: [132.48242297]
- Error of train: 0.10016
- Error of val: 97.9599

Task #4: Individual features

- Write a code that takes a list of feature 0-based ids. For each **individual** feature, build a polynomial regression with cross-features for degrees [1, 2, 3]
  - Compute train and val errors for each feature
  - Use a bar plot to **compare** the features together
- For example, assume we tried features: 0, 3, 6 and polynomial degree = 1
- Overall, generate 3 diagrams, one per polynomial degree
- Use --extra_args to pass list of comma separate Features 0-based indices
  - -1 means compare the 30 features
  - Explore: *0, 1, 2, 3, 4, 5, 6, 7, 8*
  - Explore: *6, 7, 8, 9, 10, 11, 12, 13*



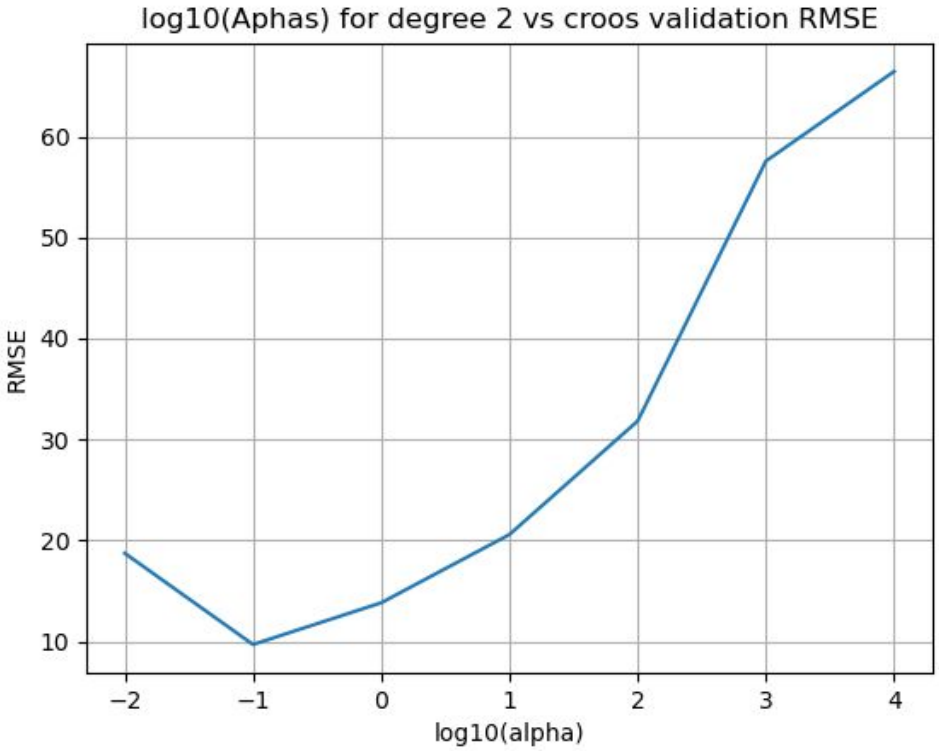
Task #4: Individual features

- For each degree in [1, 2, 3]
  - For each feature in [0, 3, 6]
    - Input: single feature
    - Train model
  - Build the graph for comparing the single feature performance

Task #5: Regularized polynomial regression

- We would like to use ridge regression with all features with polynomial degree of 2 (cross features)
- Assume we want to consider lambdas: [0.01, 0.1, 1, 10, 100, 1000, 10000]
- Use trainval data with **GridSearchCV** to find the best lambda
- Plot each lambda vs the average performance from the cross validation
  - Use cross validation with k = 4
  - Use np.log10(alphas) to easily write them on the plot
- Return your best lambda

- alphas = [0.01, 0.1, 1, 10, 100, 1000, 10000]
- Expected rmse: [18.7, 9.7, 13.8, 20.59, 31.8, 57.55, 66.4]



Task #6: Lasso Selection

- We would to use **lasso** to select features for us. Then, we will perform **ridge** on the selected features only. No polynomial features
  - model = Lasso(fit_intercept = **True**, alpha = alpha, max_iter = 10000)
- Consider these lambda for lasso selection and ridge
  - alphas = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09,
  - 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 10]
- Don't use cross-fold for lasso **selection**. First 100 for train and last 100 for test
- For each lambda:
  - Build and train a lasso model
- Find your best validation error model that has maximum of 10 selected features
  - You can learn/use SelectFromModel to give you the selected features (**get_support**())
  - Train a ridge model with the selected features (CV with above lambdas)
  - Compare with ridge that uses all the features

Organize your code

- Get use to write proper code from the begin
- No silly workarounds
- No paths hardcoding
- Well-organized code: files, functions, classes, etc
- Organize your output as one folder per a task
- Prepare a report to discuss all your insights

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Regressors Homework')

    parser.add_argument('--dataset', type=str, default='data2_200x30.csv')
    parser.add_argument('--preprocessing', type=int, default=1,
                        help='0 for no processing, 1 for min/max scaling and 2 for standrizing')

    parser.add_argument('--choice', type=int, default=6,
                        help='1 for simple linear regression, '
                             '2 for polynomial on all features with degrees [1, 2, 3, 4] with cross features'
                             '3 for polynomial on all features with degrees [1, 2, 3, 4] with monomial features'
                             '4 individual features test'
                             '5 find best lambda with grid search with ridge'
                             '6 Lasso selection'
                        )
    parser.add_argument('--extra_args', type=str, help='extra values passed', default='0,3,6')

    args = parser.parse_args()

    X, t, X_train, t_train, X_val, t_val = load_data(args.dataset)
```

Report

- Create a Google Doc report
- Discuss your different findings and learned lesson
- Which model you can use for testing purposes

Tasks: 7 - 10

- These are separate tasks, but you may use the previous dataset
- They are more about programming / OOP than ML

Task #7: Regularized Normal Equations

$$(X^T X + \lambda I)^{-1} X^T y$$

- Create class NormalEquations that:
  - Implements this regularized normal equation
  - can be use with sklearn as following:

```
for alpha in [0, 0.1, 1, 10, 100, 1000]:
    model = NormalEquations(alpha=alpha)
    #model = Ridge(alpha=alpha)

    model.fit(X_train, t_train)

    t_pred = model.predict(X_val)
    error = mean_squared_error(t_val, t_pred)

    print(model.coef_)
    print(error)
```

Task 8: Update Gradient Descent with regularization

- Introduce simple changes to convert our gradient descent code to support regularization with new hyperparameter lambda

Task 9: Grid Search for polynomial features v1

- Create a program that administrate creating a pipeline / GridSearchCV that allows learning **ridge and grid** search on:
  - **PolynomialFeatures transfer (possible degrees 1, 2, 3)**
    - Fix include_bias = False
  - Ridge alpha with possible values: [1, 0.1, 10]
  - Ridge fit_intercept with possible values: [true, false]
- Use MinMaxScaler
- We don’t care about results / data leakage.
- Just coding that

Task 10: Grid Search for polynomial features v2

- To administrate learning how to extend libraries, let's **create a class** that performs 2 steps together
  - PolynomialFeatures transfer for the values
  - Ridge model on the new features
- Process
  - The class will receive parameters for alpha, fit_intercept and ALSO the degree
  - It first, transform the data, then fit ridge to the new data
- Code usage; very close to Task 9 (so finish 9 first)
- from sklearn.base import BaseEstimator
- class PolynomialRegression(BaseEstimator)
- Next slide is optional for more support. Try to start without it

```
class PolynomialRegression(BaseEstimator):
    def __init__(self, degree = 1, **kwargs):...

    def fit(self, X, y, **kwargs):...

    def get_params(self, **kwargs):...

    def set_params(self, **kwargs):...

    def predict(self, X):...

    @property
    def coef_(self):...

    @property
    def intercept_(self):...
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."