cme250_hw2_solutions

February 25, 2019

```
In [1]: import numpy as np
        import pandas as pd
        import json
        import collections
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
```

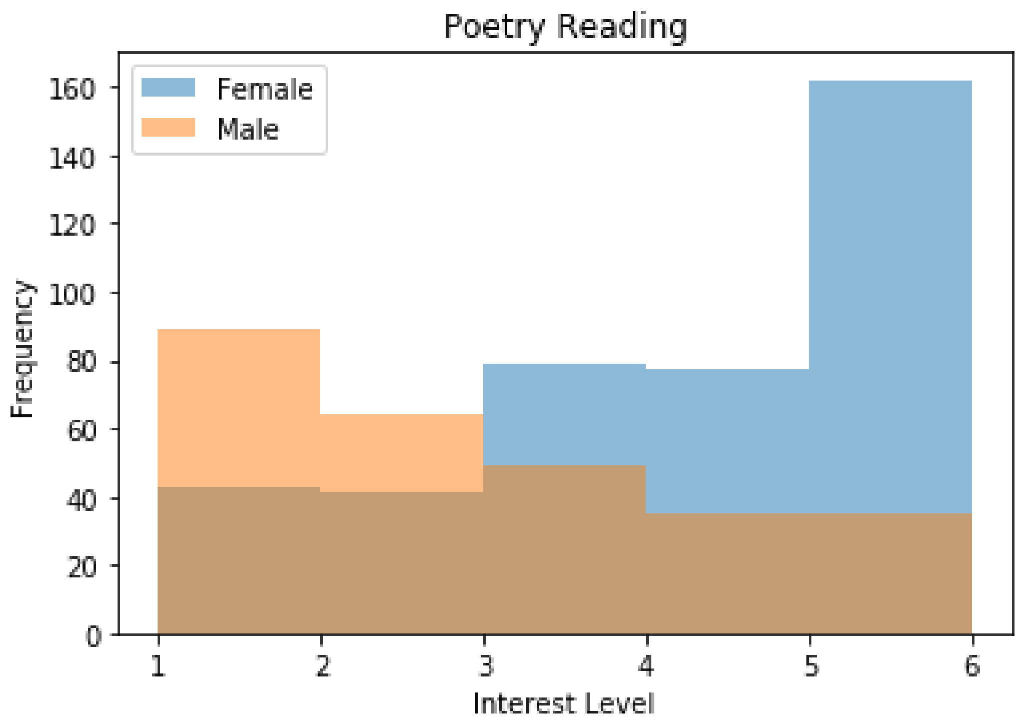
1 Part 3. Young People Survey

```
In [2]: # use pandas to read in .csv file
        path = '../data/hw2.csv'
        df = pd.read_csv(path)
```

```
In [3]: # load category-number mappings
        categories = json.load(open('../data/hw2_categorical_encodings.json'))
```

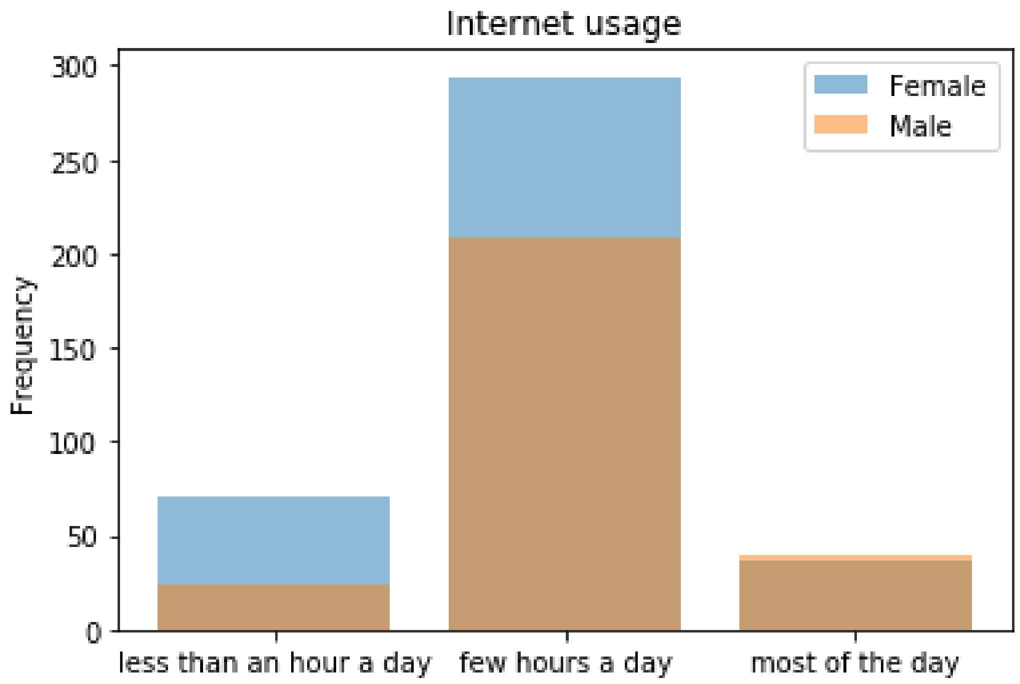
1.1 (a) Data exploration

```
In [4]: # reading histogram
        plt.hist(df[df['Gender'] == 0]['Reading'], bins=[1,2,3,4,5,6], alpha=0.5)
        plt.hist(df[df['Gender'] == 1]['Reading'], bins=[1,2,3,4,5,6], alpha=0.5)
        plt.title('Poetry Reading')
        plt.xlabel('Interest Level')
        plt.ylabel('Frequency')
        plt.legend(['Female', 'Male'])
        plt.show()
```



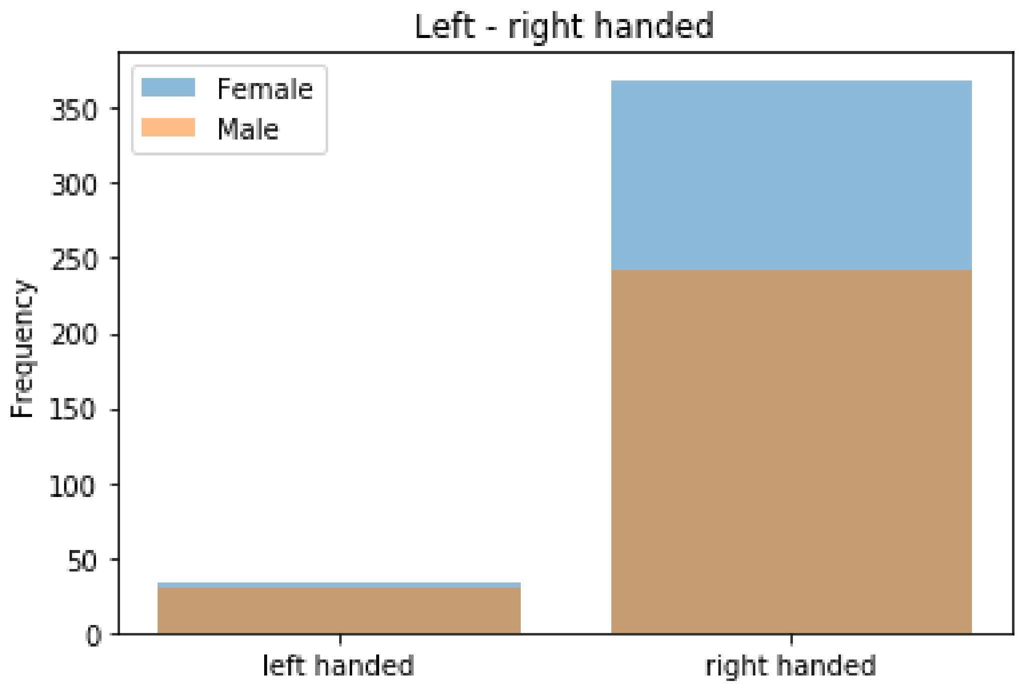
```
In [5]: # internet usage bar graph
        col = 'Internet usage'
        col_dict_female = collections.Counter(df[df['Gender'] == 0][col].values)
        col_dict_male = collections.Counter(df[df['Gender'] == 1][col].values)
        names_female = [categories[col][str(k)] for k in col_dict_female.keys()]
        names_male = [categories[col][str(k)] for k in col_dict_male.keys()]
        values_female = col_dict_female.values()
        values_male = col_dict_male.values()

        plt.bar(names_female, values_female, alpha=0.5)
        plt.bar(names_male, values_male, alpha=0.5)
        plt.title(col)
        plt.ylabel('Frequency')
        plt.legend(['Female', 'Male'])
        plt.show()
```



```
In [6]: # handedness bar graph
        col = 'Left - right handed'
        col_dict_female = collections.Counter(df[df['Gender'] == 0][col].values)
        col_dict_male = collections.Counter(df[df['Gender'] == 1][col].values)
        names_female = [categories[col][str(k)] for k in col_dict_female.keys()]
        names_male = [categories[col][str(k)] for k in col_dict_male.keys()]
        values_female = col_dict_female.values()
        values_male = col_dict_male.values()

        plt.bar(names_female, values_female, alpha=0.5)
        plt.bar(names_male, values_male, alpha=0.5)
        plt.title(col)
        plt.ylabel('Frequency')
        plt.legend(['Female', 'Male'])
        plt.show()
```



1.2 (b) Dataset split

```
In [5]: X = df.drop('Gender', axis=1)
        y = df['Gender']

        X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, random_state=0)
        X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, random_state=0)
```

```
In [6]: print("Training set dimensions: X is {}, y is {}".format(X_train.shape, y_train.shape))
        print("Validation set dimensions: X is {}, y is {}".format(X_val.shape, y_val.shape))
        print("Test set dimensions: X is {}, y is {}".format(X_test.shape, y_test.shape))
```

```
Training set dimensions: X is (378, 149), y is (378,)
Validation set dimensions: X is (127, 149), y is (127,)
Test set dimensions: X is (169, 149), y is (169,)
```

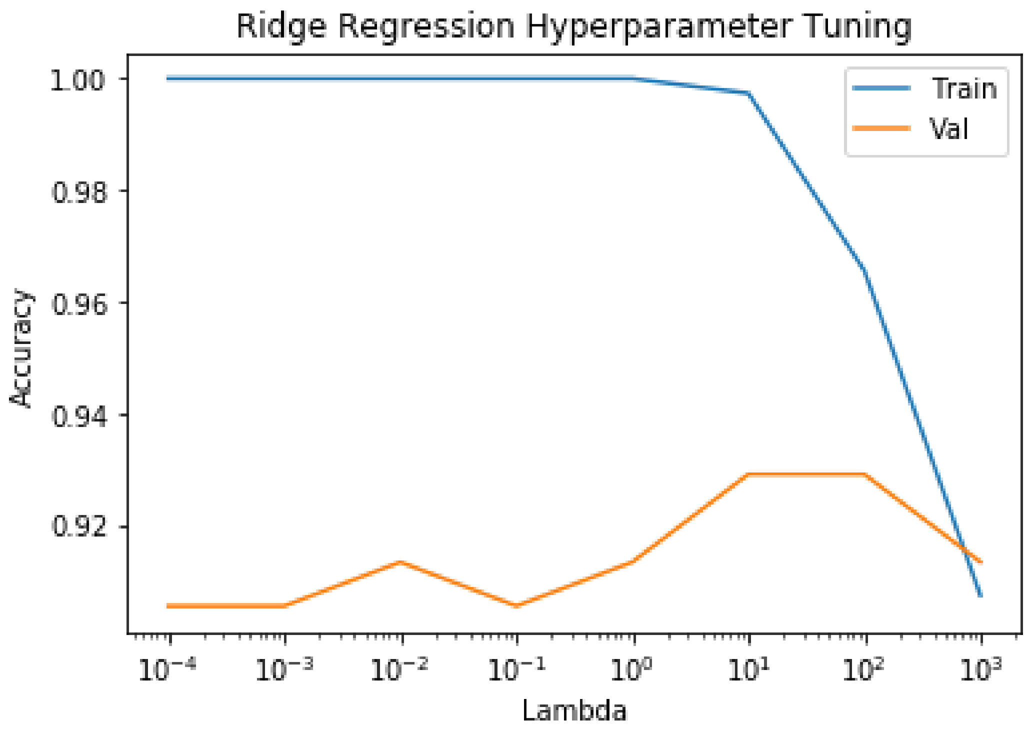
1.3 (c) Logistic regression with ridge penalty

```
In [7]: # define hyperparameter search space and vectors to house results
        lambdas = np.logspace(-4, 3, 8)
        ridge_acc_train = np.zeros(len(lambdas))
        ridge_acc_val = np.zeros(len(lambdas))

        # iterate over lambdas and train an l2-penalized logistic regression for each lambda
        for i, l in enumerate(lambdas):
            ridge = LogisticRegression(penalty='l2', C=1./l)
            ridge.fit(X_train, y_train)

            ridge_acc_train[i] = ridge.score(X_train, y_train)
            ridge_acc_val[i] = ridge.score(X_val, y_val)
```

```
In [8]: # plot accuracy vs. lambda for train and validation sets
        plt.semilogx(lambdas, ridge_acc_train)
        plt.semilogx(lambdas, ridge_acc_val)
        plt.xlabel('Lambda')
        plt.ylabel('Accuracy')
        plt.title('Ridge Regression Hyperparameter Tuning')
        plt.legend(['Train', 'Val'])
        plt.show()
```



From the graph, we can see that the highest accuracies were achieved for $\lambda = 10$ and $\lambda = 100$.

```
In [9]: ridge = LogisticRegression(penalty='l2', C=1./10)
        ridge.fit(X_train, y_train)
        print("Number of coefficients exactly zero: {}".format(np.sum(ridge.coef_ == 0)))

Number of coefficients exactly zero: 0
```

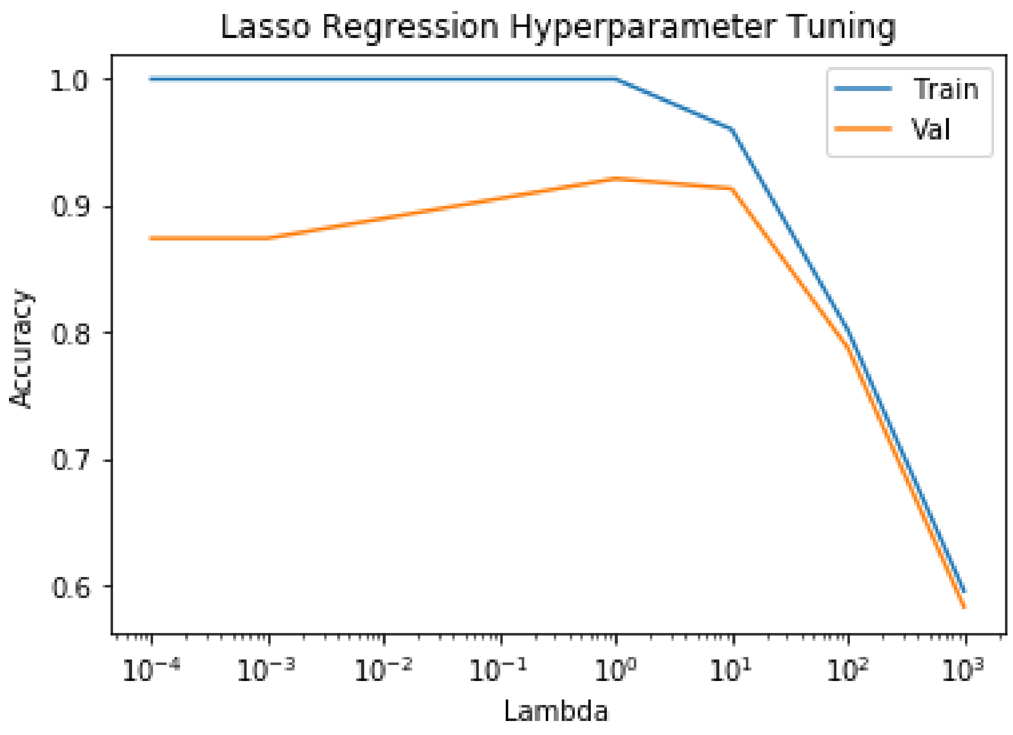
1.4 (d) Logistic regression with lasso penalty

```
In [10]: # define hyperparameter search space and vectors to house results
         lambdas = np.logspace(-4, 3, 8)
         lasso_acc_train = np.zeros(len(lambdas))
         lasso_acc_val = np.zeros(len(lambdas))

         # iterate over lambdas and train an l1-penalized logistic regression for each lambda
         for i, l in enumerate(lambdas):
             lasso = LogisticRegression(penalty='l1', C=1./l)
             lasso.fit(X_train, y_train)
```

```
            lasso_acc_train[i] = lasso.score(X_train, y_train)
            lasso_acc_val[i] = lasso.score(X_val, y_val)
```

```
In [11]: # plot accuracy vs. lambda for train and validation sets
         plt.semilogx(lambdas, lasso_acc_train)
         plt.semilogx(lambdas, lasso_acc_val)
         plt.xlabel('Lambda')
         plt.ylabel('Accuracy')
         plt.title('Lasso Regression Hyperparameter Tuning')
         plt.legend(['Train', 'Val'])
         plt.show()
```



From the graph, we can see that the highest accuracy was achieved for $\lambda = 1$.

```
In [12]: lasso = LogisticRegression(penalty='l1', C=1.)
         lasso.fit(X_train, y_train)
         print("Number of coefficients exactly zero: {}".format(np.sum(lasso.coef_ == 0)))
```

```
Number of coefficients exactly zero: 82
```

1.5 (e) Generalization error

```
In [13]: lasso = LogisticRegression(penalty='l1', C=1.)
         lasso.fit(X_trainval, y_trainval)
         y_pred = lasso.predict(X_test)
         print("Test set accuracy: {:0.4f}".format(accuracy_score(y_test, y_pred)))
```

```
Test set accuracy: 0.9112
```

```
In [ ]:
```