hw2

May 5, 2020

[1]:
```
import numpy as np
import pathlib
```

Data import

[5]:
```
DATA_DIR = pathlib.Path("../hw1/hw1-data")

train_path = DATA_DIR / "income.train.txt.5k"
eval_path = DATA_DIR / "income.dev.txt"
test_path = DATA_DIR / "income.test.blind"
train_and_eval_path = DATA_DIR / "income.combined.6k"

COL_NAMES = [
    "age",
    "sector",
    "education",
    "marital-status",
    "occupation",
    "race",
    "gender",
    "hours-per-week",
    "country-of-origin",
    "target"
]
```

[6]:
```
def txt_file_to_df(file_path):

    DELIMITER = ","

    def parse_row(row, delimiter=DELIMITER):
        cells = row.split(delimiter)
        parsed_row = [
            int(cell.strip())
            if cell.isnumeric()
            else cell.strip()
            for cell in cells
        ]
```

```
        return parsed_row

    data = []
    with open(file_path) as in_:
        raw_rows = in_.readlines()

    for row in raw_rows:
        parsed_row = parse_row(row)
        data.append(parsed_row)

    df = {col: val for col, val in zip(*[COL_NAMES, zip(*data)])}
    return df, data
```

[7]:
```
# First person in eval
df, data = txt_file_to_df(train_path)
data[0], df.keys()
```

[7]:
```
([50,
  'Self-emp-not-inc',
  'Bachelors',
  'Married-civ-spouse',
  'Exec-managerial',
  'White',
  'Male',
  '13',
  'United-States',
  '<=50K'],
 dict_keys(['age', 'sector', 'education', 'marital-status', 'occupation',
'race', 'gender', 'hours-per-week', 'country-of-origin', 'target']))
```

[56]:
```
'''
Takes file name and trained encoders as input to
load, parse and transform dataset into modelling ready dataframe
'''

def txt_to_encoded_df(file_path, encoders):

    # Load and parse data
    df, _ = txt_file_to_df(file_path)

    # Encode and transform each col
    encoded_df = []
    for col in df.keys():

        if col == "target":
            continue
```

```
        elif col in encoders:
            encoder = encoders[col]
            col_values = encoder.transform(df[col])

        else:
            col_values = np.array(df[col], dtype=np.float64).
↪reshape(len(df[col]), -1) / 50.

        encoded_df.append(col_values)

    # Make a flat dataset from all cols
    encoded_df = np.hstack(encoded_df)

    if "test" in str(file_path):
        return encoded_df

    return encoded_df, df["target"]
```

[57]:
```
# One hot encoder

class OneHotEncoder(object):

    def __init__(self):
        pass

    def fit(self, column):
        self.unique_categories = list(set(column))
        self.catg_index = {catg: cid for cid, catg in enumerate(self.
↪unique_categories)}

    def transform(self, column):
        ct, ck = 0, []
        self.ohe_column = np.zeros((len(column), len(self.unique_categories)),␣
↪dtype=np.float64)
        for i, catg in enumerate(column):
            try:
                j = self.catg_index[catg]
                self.ohe_column[i, j] = 1
            except KeyError:
                ct += 1
                ck.append(catg)

        print(f'Failed {ct} times because of {ck}')
        return self.ohe_column

    def fit_transform(self, column):
        self.fit(column=column)
```

```
        return self.transform(column=column)
```

[58]:
```
# One hot encoding categorical vars

def make_encoders(categorical_columns, train_df):
    '''
    Takes categorical column names and df to make encoder objects using data
    '''

    catg_columns = categorical_columns.keys()

    # Init encoder objects
    col_encoders = {col: OneHotEncoder() for col in catg_columns}

    # Train encoders
    for col in catg_columns:
        col_values = col_encoders[col].fit(train_df[col])

    return col_encoders


cols_catg = {
    "sector": 7,
    "education": 16,
    "marital-status": 4,
    "occupation": 14,
    "race": 5,
    "gender": 2,
    "country-of-origin": 39,
    "hours-per-week": "*",
    "age": "*"
}

combined_df, _ = txt_file_to_df(train_and_eval_path)

make_encoders(categorical_columns=cols_catg, train_df=combined_df)
```

[58]:
```
{'sector': <__main__.OneHotEncoder at 0x1216bdcd0>,
 'education': <__main__.OneHotEncoder at 0x121723b90>,
 'marital-status': <__main__.OneHotEncoder at 0x1216d1a50>,
 'occupation': <__main__.OneHotEncoder at 0x123fc8510>,
 'race': <__main__.OneHotEncoder at 0x123fe3050>,
 'gender': <__main__.OneHotEncoder at 0x1241d3390>,
 'country-of-origin': <__main__.OneHotEncoder at 0x124262910>,
 'hours-per-week': <__main__.OneHotEncoder at 0x12426eed0>,
 'age': <__main__.OneHotEncoder at 0x1216eb650>}
```

[16]:
```
# Example of this dataset's ETL cycle

combined_df_raw, _ = txt_file_to_df(train_and_eval_path)
one_hot_encoders = make_encoders(categorical_columns=cols_catg,␣
 ↪train_df=combined_df_raw)
train_df = txt_to_encoded_df(file_path=train_path, encoders=one_hot_encoders)

print(train_df.shape)
train_df
```

```
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
(5000, 233)
```

[16]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

Data ETL

[255]:
```
combined_df_raw, _ = txt_file_to_df(train_and_eval_path)
one_hot_encoders = make_encoders(categorical_columns=cols_catg,␣
 ↪train_df=combined_df_raw)

train_df = txt_file_to_df(train_path)[0]
eval_df = txt_file_to_df(eval_path)[0]
test_df = txt_file_to_df(test_path)[0]

train_X, train_y = txt_to_encoded_df(file_path=train_path,␣
 ↪encoders=one_hot_encoders)
eval_X, eval_y = txt_to_encoded_df(file_path=eval_path,␣
 ↪encoders=one_hot_encoders)
test_X = txt_to_encoded_df(file_path=test_path, encoders=one_hot_encoders)
```

```
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
```

```
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 1 times because of [83]
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 0 times because of []
Failed 2 times because of ['31', '29']
Failed 0 times because of []
```

0.1 Feature Map

0.1.1 Why binarize numerical fields?

KNN uses distance metric while Perceptron uses dot product. Binarizing age would help optimization algorithm to preserve which ages are "useful" while updating W. This usefulness would be measured by dot product. If we hadn't binarized, dot product won't have this granular distingushing power but more liekly saturate.

0.1.2 Total number of features?

234 Including bias and a couple of feature that are only present in dev set

0.2 Perceptron and Averaged Perceptron

0.2.1 Basic perceptron algorithm

[110]:
```
class Perceptron(object):

    def __init__(self, epochs, n_features):
        self.n_epochs = epochs
        self.n_features = n_features

        # Integrating bias into W
```

```
        self.W = np.zeros(self.n_features + 1)

    def fit(self, X, y):
        self.n_examples = X.shape[0]

        updates = 0
        for rid in range(self.n_examples):
            row = X[rid, :]
            target = y[rid]

            if np.dot(self.W, row) * target <= 0:
                self.W += target * row
                updates += 1

        return updates

    @staticmethod
    def binarize_y(y):
        y_bin = np.array(y)
        y_bin[y_bin == "<=50K"] = -1
        y_bin[y_bin != "-1"] = 1
        y_bin = y_bin.astype(np.int64)

        return y_bin

    @staticmethod
    def expand_bias_dim(X):
        bias_dim = np.ones((X.shape[0], 1))

        return np.hstack((bias_dim, X))



    def evaluate(self, teX, tey):
        predictions = (np.dot(teX, self.W) > 0).astype(np.float)
        predictions[predictions == 0] = -1

        err_rate = np.mean(predictions != tey)
        pos_rate = np.mean(predictions == 1)

        return err_rate, pos_rate

    def summary(self, trX, trY, teX, teY):

        trX = self.expand_bias_dim(trX)
        trYb = self.binarize_y(trY)
        teX = self.expand_bias_dim(teX)
```

```
        teYb = self.binarize_y(teY)

        for e in range(1, self.n_epochs + 1):

            n_updates = self.fit(trX, trYb)
            perc_updates = n_updates * 100 / self.n_examples

            err_rate, pos_rate = self.evaluate(teX, teYb)
            err_perc, pos_perc = err_rate * 100, pos_rate * 100

            print(
            f'epoch {e} updates {n_updates} ({perc_updates:.2f}%) dev_error␣
↪{err_perc:.2f}% (+:{pos_perc:.2f}%)')
```

[111]:
```
pt = Perceptron(epochs=5, n_features=train_X.shape[1])
pt.summary(trX=train_X, trY=train_y, teX=eval_X, teY=eval_y)
```

```
epoch 1 updates 1257 (25.14%) dev_error 21.10% (+:27.50%)
epoch 2 updates 1221 (24.42%) dev_error 18.80% (+:25.40%)
epoch 3 updates 1177 (23.54%) dev_error 17.50% (+:21.50%)
epoch 4 updates 1170 (23.40%) dev_error 19.10% (+:12.30%)
epoch 5 updates 1172 (23.44%) dev_error 18.70% (+:17.70%)
```

0.2.2 Averaged perceptron algorithm

[112]:
```
class AvgPerceptron(object):

    def __init__(self, epochs, n_features):
        self.n_epochs = epochs
        self.n_features = n_features

        # Integrating bias into W
        self.W = np.zeros(self.n_features + 1)

        self.av_W = np.zeros(self.n_features + 1)
        self.c = 0

    def fit(self, X, y):
        self.n_examples = X.shape[0]

        updates = 0
        for rid in range(self.n_examples):
            row = X[rid, :]
            target = y[rid]

            if np.dot(self.W, row) * target <= 0:
                self.W += target * row
```

```
                self.av_W += self.c * target * row
                updates += 1

            self.c += 1

        return updates

    @staticmethod
    def binarize_y(y):
        y_bin = np.array(y)
        y_bin[y_bin == "<=50K"] = -1
        y_bin[y_bin != "-1"] = 1
        y_bin = y_bin.astype(np.int64)

        return y_bin

    @staticmethod
    def expand_bias_dim(X):
        bias_dim = np.ones((X.shape[0], 1))

        return np.hstack((bias_dim, X))



    def evaluate(self, teX, tey):

        effective_W = self.c * self.W - self.av_W
        predictions = (np.dot(teX, effective_W) > 0).astype(np.float)
        predictions[predictions == 0] = -1

        err_rate = np.mean(predictions != tey)
        pos_rate = np.mean(predictions == 1)

        return err_rate, pos_rate

    def summary(self, trX, trY, teX, teY):

        trX = self.expand_bias_dim(trX)
        trYb = self.binarize_y(trY)
        teX = self.expand_bias_dim(teX)
        teYb = self.binarize_y(teY)

        for e in range(1, self.n_epochs + 1):

            n_updates = self.fit(trX, trYb)
            perc_updates = n_updates * 100 / self.n_examples
```

```
            err_rate, pos_rate = self.evaluate(teX, teYb)
            err_perc, pos_perc = err_rate * 100, pos_rate * 100

            print(
            f'epoch {e} updates {n_updates} ({perc_updates:.2f}%) dev_error␣
↪{err_perc:.2f}% (+:{pos_perc:.2f}%)')
```

[113]:
```
apt = AvgPerceptron(epochs=5, n_features=train_X.shape[1])
apt.summary(trX=train_X, trY=train_y, teX=eval_X, teY=eval_y)
```

```
epoch 1 updates 1257 (25.14%) dev_error 15.00% (+:18.60%)
epoch 2 updates 1221 (24.42%) dev_error 15.10% (+:19.30%)
epoch 3 updates 1177 (23.54%) dev_error 14.80% (+:20.00%)
epoch 4 updates 1170 (23.40%) dev_error 14.70% (+:19.30%)
epoch 5 updates 1172 (23.44%) dev_error 14.80% (+:20.00%)
```

0.2.3 Observations:

Vanilla's updates seemed jumpy while Averaged Perceptron updates are much stabler. Also the dev error rate of AP is smaller at 14.70% compared to vanilla's 17.50%

0.2.4 Positive & Negative features

Positives Married Education: doctorate, prof-school Country: Iran

Negatives Ages: 26, 28 Education: 7-8th Occupation: Farming-Fishing

0.2.5 Male, Female

These features present in every example (both positive & negative) and would add +1 during dot product. To center the dot product around 0, these may have drifted towards negative values?

0.2.6 Bias

[239]:
```
apt.W[0]
```

[239]: -9.0

0.2.7 Is the update % above equivalent to "training error"?

No, the update % is simply number of misclassifed examples on the rolling basis from previous epoch. The weight vector changes as we perform updates leaving no bechmark to calculate training error that would be consistent over any arrangement of inputs

0.3 Comparing Perceptron with KNN

0.3.1 What are the major advantages of perceptron over k-NN?

- Perceptron summarizes training data into weight vector which makes inference much faster (as KNN has to compare over all training data to classify when Perceptron has to compare

over just W)
- Perceptron uses "dot product" while KNN uses "distance metric": this translates to KNN attributing importance to vectors which may be completely orthogonal to dataset "prime" vector but are still equidistant.

0.3.2 Design and execute a small experiment to demonstrate your point(s).

- KNN evaluation time: 5 seconds (k=41)
- Perceptron evaluation time: ~<1 seconds

Perceptron inference is much faster!

0.4 Experiments

0.4.1 Reordering training data for positives to come first

Yes, both models degraded as they don't get to correct errors until they exhaust all positive examples. They are now as good as just trained on negatives.

[125]:
```
# otrain_X, otrain_y = train_X, train_y

sorted_idx = np.argpartition(train_y, range(len(train_y)))[::-1]
ordered_train_X = train_X[sorted_idx]
ordered_train_y = np.array(train_y)[sorted_idx].tolist()
```

[132]:
```
pt = Perceptron(epochs=50, n_features=ordered_train_X.shape[1])
pt.summary(trX=ordered_train_X, trY=ordered_train_y, teX=eval_X, teY=eval_y)
```

```
epoch 1 updates 4 (0.08%) dev_error 23.60% (+:0.00%)
epoch 2 updates 6 (0.12%) dev_error 23.60% (+:0.00%)
epoch 3 updates 6 (0.12%) dev_error 23.60% (+:0.00%)
epoch 4 updates 9 (0.18%) dev_error 23.60% (+:0.00%)
epoch 5 updates 10 (0.20%) dev_error 23.60% (+:0.00%)
epoch 6 updates 11 (0.22%) dev_error 23.60% (+:0.00%)
epoch 7 updates 14 (0.28%) dev_error 23.60% (+:0.00%)
epoch 8 updates 14 (0.28%) dev_error 23.60% (+:0.00%)
epoch 9 updates 15 (0.30%) dev_error 23.60% (+:0.00%)
epoch 10 updates 16 (0.32%) dev_error 23.60% (+:0.00%)
epoch 11 updates 16 (0.32%) dev_error 23.60% (+:0.00%)
epoch 12 updates 16 (0.32%) dev_error 23.60% (+:0.00%)
epoch 13 updates 18 (0.36%) dev_error 23.60% (+:0.00%)
epoch 14 updates 17 (0.34%) dev_error 23.60% (+:0.00%)
epoch 15 updates 17 (0.34%) dev_error 23.60% (+:0.00%)
epoch 16 updates 18 (0.36%) dev_error 23.60% (+:0.00%)
epoch 17 updates 19 (0.38%) dev_error 23.60% (+:0.00%)
epoch 18 updates 18 (0.36%) dev_error 23.60% (+:0.00%)
epoch 19 updates 18 (0.36%) dev_error 23.60% (+:0.00%)
epoch 20 updates 20 (0.40%) dev_error 23.60% (+:0.00%)
epoch 21 updates 21 (0.42%) dev_error 23.60% (+:0.00%)
epoch 22 updates 22 (0.44%) dev_error 23.60% (+:0.00%)
```

```
epoch 23 updates 22 (0.44%) dev_error 23.60% (+:0.00%)
epoch 24 updates 20 (0.40%) dev_error 23.60% (+:0.00%)
epoch 25 updates 21 (0.42%) dev_error 23.60% (+:0.00%)
epoch 26 updates 19 (0.38%) dev_error 23.60% (+:0.00%)
epoch 27 updates 22 (0.44%) dev_error 23.60% (+:0.00%)
epoch 28 updates 22 (0.44%) dev_error 23.60% (+:0.00%)
epoch 29 updates 20 (0.40%) dev_error 23.60% (+:0.00%)
epoch 30 updates 23 (0.46%) dev_error 23.60% (+:0.00%)
epoch 31 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 32 updates 23 (0.46%) dev_error 23.60% (+:0.00%)
epoch 33 updates 21 (0.42%) dev_error 23.60% (+:0.00%)
epoch 34 updates 22 (0.44%) dev_error 23.50% (+:0.10%)
epoch 35 updates 23 (0.46%) dev_error 23.60% (+:0.00%)
epoch 36 updates 22 (0.44%) dev_error 23.50% (+:0.10%)
epoch 37 updates 25 (0.50%) dev_error 23.60% (+:0.00%)
epoch 38 updates 26 (0.52%) dev_error 23.60% (+:0.00%)
epoch 39 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 40 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 41 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 42 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 43 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 44 updates 24 (0.48%) dev_error 23.60% (+:0.00%)
epoch 45 updates 25 (0.50%) dev_error 23.60% (+:0.00%)
epoch 46 updates 28 (0.56%) dev_error 23.60% (+:0.00%)
epoch 47 updates 26 (0.52%) dev_error 23.60% (+:0.00%)
epoch 48 updates 25 (0.50%) dev_error 23.60% (+:0.00%)
epoch 49 updates 26 (0.52%) dev_error 23.50% (+:0.10%)
epoch 50 updates 27 (0.54%) dev_error 23.50% (+:0.10%)
```

[131]:
```
apt = AvgPerceptron(epochs=50, n_features=ordered_train_X.shape[1])
apt.summary(trX=ordered_train_X, trY=ordered_train_y, teX=eval_X, teY=eval_y)
```

```
epoch 1 updates 4 (0.08%) dev_error 23.50% (+:0.10%)
epoch 2 updates 6 (0.12%) dev_error 23.60% (+:0.00%)
epoch 3 updates 6 (0.12%) dev_error 23.70% (+:0.10%)
epoch 4 updates 9 (0.18%) dev_error 23.70% (+:0.30%)
epoch 5 updates 10 (0.20%) dev_error 23.50% (+:0.10%)
epoch 6 updates 11 (0.22%) dev_error 23.40% (+:0.20%)
epoch 7 updates 14 (0.28%) dev_error 23.50% (+:0.30%)
epoch 8 updates 14 (0.28%) dev_error 23.60% (+:0.20%)
epoch 9 updates 15 (0.30%) dev_error 23.40% (+:0.40%)
epoch 10 updates 16 (0.32%) dev_error 23.30% (+:0.50%)
epoch 11 updates 16 (0.32%) dev_error 23.20% (+:0.60%)
epoch 12 updates 16 (0.32%) dev_error 23.20% (+:1.00%)
epoch 13 updates 18 (0.36%) dev_error 23.10% (+:1.30%)
epoch 14 updates 17 (0.34%) dev_error 23.30% (+:1.30%)
epoch 15 updates 17 (0.34%) dev_error 23.10% (+:1.70%)
epoch 16 updates 18 (0.36%) dev_error 23.10% (+:1.70%)
```

```
epoch 17 updates 19 (0.38%) dev_error 22.90% (+:1.90%)
epoch 18 updates 18 (0.36%) dev_error 22.80% (+:2.00%)
epoch 19 updates 18 (0.36%) dev_error 22.90% (+:2.10%)
epoch 20 updates 20 (0.40%) dev_error 22.70% (+:2.30%)
epoch 21 updates 21 (0.42%) dev_error 22.80% (+:2.20%)
epoch 22 updates 22 (0.44%) dev_error 22.80% (+:2.20%)
epoch 23 updates 22 (0.44%) dev_error 22.80% (+:2.20%)
epoch 24 updates 20 (0.40%) dev_error 22.70% (+:2.30%)
epoch 25 updates 21 (0.42%) dev_error 22.80% (+:2.20%)
epoch 26 updates 19 (0.38%) dev_error 22.80% (+:2.40%)
epoch 27 updates 22 (0.44%) dev_error 22.80% (+:2.40%)
epoch 28 updates 22 (0.44%) dev_error 22.70% (+:2.50%)
epoch 29 updates 20 (0.40%) dev_error 22.70% (+:2.70%)
epoch 30 updates 23 (0.46%) dev_error 22.70% (+:2.90%)
epoch 31 updates 24 (0.48%) dev_error 22.90% (+:2.90%)
epoch 32 updates 23 (0.46%) dev_error 22.80% (+:3.00%)
epoch 33 updates 21 (0.42%) dev_error 22.80% (+:3.00%)
epoch 34 updates 22 (0.44%) dev_error 22.80% (+:3.00%)
epoch 35 updates 23 (0.46%) dev_error 22.80% (+:3.00%)
epoch 36 updates 22 (0.44%) dev_error 22.80% (+:3.00%)
epoch 37 updates 25 (0.50%) dev_error 22.80% (+:3.20%)
epoch 38 updates 26 (0.52%) dev_error 22.80% (+:3.20%)
epoch 39 updates 24 (0.48%) dev_error 22.80% (+:3.20%)
epoch 40 updates 24 (0.48%) dev_error 23.10% (+:3.50%)
epoch 41 updates 24 (0.48%) dev_error 23.20% (+:3.60%)
epoch 42 updates 24 (0.48%) dev_error 23.40% (+:3.80%)
epoch 43 updates 24 (0.48%) dev_error 23.50% (+:3.90%)
epoch 44 updates 24 (0.48%) dev_error 23.50% (+:4.10%)
epoch 45 updates 25 (0.50%) dev_error 23.30% (+:4.30%)
epoch 46 updates 28 (0.56%) dev_error 23.30% (+:4.30%)
epoch 47 updates 26 (0.52%) dev_error 23.20% (+:4.20%)
epoch 48 updates 25 (0.50%) dev_error 23.20% (+:4.20%)
epoch 49 updates 26 (0.52%) dev_error 23.30% (+:4.30%)
epoch 50 updates 27 (0.54%) dev_error 23.30% (+:4.30%)
```

0.4.2 Feature Engineering

Appending numerical features as-is

[160]:
```
tr_age = np.array(train_df["age"]).reshape(-1, 1)
te_age = np.array(eval_df["age"]).reshape(-1, 1)
tr_hw_per_week = np.array(train_df["hours-per-week"]).astype(np.int).
 ↪reshape(-1, 1)
te_hw_per_week = np.array(eval_df["hours-per-week"]).astype(np.int).reshape(-1,␣
 ↪1)

nappended_train_X = np.hstack((train_X, tr_age, tr_hw_per_week))
nappended_test_X = np.hstack((test_X, te_age, te_hw_per_week))
```

[163]:
```
pt = Perceptron(epochs=5, n_features=nappended_train_X.shape[1])
pt.summary(trX=nappended_train_X, trY=train_y, teX=nappended_test_X, teY=eval_y)
```

```
epoch 1 updates 1858 (37.16%) dev_error 23.80% (+:0.20%)
epoch 2 updates 1676 (33.52%) dev_error 23.80% (+:0.20%)
epoch 3 updates 1601 (32.02%) dev_error 36.20% (+:23.40%)
epoch 4 updates 1516 (30.32%) dev_error 37.50% (+:25.90%)
epoch 5 updates 1510 (30.20%) dev_error 26.50% (+:3.90%)
```

[164]:
```
apt = AvgPerceptron(epochs=5, n_features=nappended_train_X.shape[1])
apt.summary(trX=nappended_train_X, trY=train_y, teX=nappended_test_X,␣
↪teY=eval_y)
```

```
epoch 1 updates 1858 (37.16%) dev_error 23.60% (+:0.00%)
epoch 2 updates 1676 (33.52%) dev_error 23.70% (+:0.10%)
epoch 3 updates 1601 (32.02%) dev_error 24.80% (+:1.20%)
epoch 4 updates 1516 (30.32%) dev_error 26.00% (+:3.00%)
epoch 5 updates 1510 (30.20%) dev_error 27.70% (+:5.70%)
```

Centering each numerical dimension

[166]:
```
tr_age = np.array(train_df["age"]).reshape(-1, 1)
tr_age = tr_age - np.mean(tr_age)
te_age = np.array(eval_df["age"]).reshape(-1, 1)
te_age = te_age - np.mean(te_age)

tr_hw_per_week = np.array(train_df["hours-per-week"]).astype(np.int).
↪reshape(-1, 1)
tr_hw_per_week = tr_hw_per_week - np.mean(tr_hw_per_week)
te_hw_per_week = np.array(eval_df["hours-per-week"]).astype(np.int).reshape(-1,␣
↪1)
te_hw_per_week = te_hw_per_week - np.mean(te_hw_per_week)

cn_nappended_train_X = np.hstack((train_X, tr_age, tr_hw_per_week))
cn_nappended_test_X = np.hstack((test_X, te_age, te_hw_per_week))
```

[177]:
```
pt = Perceptron(epochs=5, n_features=nappended_train_X.shape[1])
pt.summary(trX=cn_nappended_train_X, trY=train_y, teX=cn_nappended_test_X,␣
↪teY=eval_y)
```

```
epoch 1 updates 1183 (23.66%) dev_error 36.10% (+:34.50%)
epoch 2 updates 1144 (22.88%) dev_error 29.10% (+:14.70%)
epoch 3 updates 1139 (22.78%) dev_error 27.20% (+:14.20%)
epoch 4 updates 1127 (22.54%) dev_error 33.70% (+:23.30%)
epoch 5 updates 1117 (22.34%) dev_error 29.90% (+:16.70%)
```

[178]:
```
apt = AvgPerceptron(epochs=5, n_features=nappended_train_X.shape[1])
```

```
apt.summary(trX=cn_nappended_train_X, trY=train_y, teX=cn_nappended_test_X,␣
↪teY=eval_y)
```

```
epoch 1 updates 1183 (23.66%) dev_error 31.10% (+:16.90%)
epoch 2 updates 1144 (22.88%) dev_error 31.70% (+:17.70%)
epoch 3 updates 1139 (22.78%) dev_error 31.80% (+:17.80%)
epoch 4 updates 1127 (22.54%) dev_error 32.40% (+:18.20%)
epoch 5 updates 1117 (22.34%) dev_error 32.60% (+:18.80%)
```

Normalizing numerical features

[174]:
```
tr_age = np.array(train_df["age"]).reshape(-1, 1)
tr_age = tr_age - np.mean(tr_age)
tr_age = tr_age / np.std(tr_age)
te_age = np.array(eval_df["age"]).reshape(-1, 1)
te_age = te_age - np.mean(te_age)
te_age = te_age / np.std(te_age)

tr_hw_per_week = np.array(train_df["hours-per-week"]).astype(np.int).
↪reshape(-1, 1)
tr_hw_per_week = tr_hw_per_week - np.mean(tr_hw_per_week)
tr_hw_per_week = tr_hw_per_week / np.std(tr_hw_per_week)
te_hw_per_week = np.array(eval_df["hours-per-week"]).astype(np.int).reshape(-1,␣
↪1)
te_hw_per_week = te_hw_per_week - np.mean(te_hw_per_week)
te_hw_per_week = te_hw_per_week / np.std(te_hw_per_week)

nr_nappended_train_X = np.hstack((train_X, tr_age, tr_hw_per_week))
nr_nappended_test_X = np.hstack((test_X, te_age, te_hw_per_week))
```

[175]:
```
pt = Perceptron(epochs=5, n_features=nappended_train_X.shape[1])
pt.summary(trX=nr_nappended_train_X, trY=train_y, teX=nr_nappended_test_X,␣
↪teY=eval_y)
```

```
epoch 1 updates 1183 (23.66%) dev_error 36.10% (+:34.50%)
epoch 2 updates 1144 (22.88%) dev_error 29.10% (+:14.70%)
epoch 3 updates 1139 (22.78%) dev_error 27.20% (+:14.20%)
epoch 4 updates 1127 (22.54%) dev_error 33.70% (+:23.30%)
epoch 5 updates 1117 (22.34%) dev_error 29.90% (+:16.70%)
```

[176]:
```
apt = AvgPerceptron(epochs=5, n_features=nappended_train_X.shape[1])
apt.summary(trX=nr_nappended_train_X, trY=train_y, teX=nr_nappended_test_X,␣
↪teY=eval_y)
```

```
epoch 1 updates 1183 (23.66%) dev_error 31.10% (+:16.90%)
epoch 2 updates 1144 (22.88%) dev_error 31.70% (+:17.70%)
epoch 3 updates 1139 (22.78%) dev_error 31.80% (+:17.80%)
epoch 4 updates 1127 (22.54%) dev_error 32.40% (+:18.20%)
```

```
epoch 5 updates 1117 (22.34%) dev_error 32.60% (+:18.80%)
```

Adding some binary combination features

[257]:
```python
def make_binary_combinations(fa, fb, df):
    fa_enco = one_hot_encoders[fa]
    fb_enco = one_hot_encoders[fb]
    fa_catg = fa_enco.catg_index
    fb_catg = fb_enco.catg_index
    n_fa_catg = len(fa_catg)
    n_fb_catg = len(fb_catg)


    enum = 0
    fa_vals, fb_vals = df[fa], df[fb]
    combinations = np.zeros((len(fa_vals), n_fa_catg * n_fb_catg))

    error_count = 0
    for fai, fbi in zip(*[fa_vals, fb_vals]):
        try:
            aidx = fa_catg[fai]
            bidx = fb_catg[fbi]
        except KeyError:
            error_count += 1

        linear_idx = n_fb_catg * aidx + bidx
        combinations[enum, linear_idx] = 1
        enum += 1

    print(f'Failed computing binary combinations for {error_count} examples.')

    return combinations
```

[232]:
```python
from itertools import combinations

BIN_COMBOS = [
    "age",
    "sector",
    "education",
    "marital-status",
    "occupation",
    "race",
    "gender",
    "hours-per-week",
    "country-of-origin",
#     "target"
]
```

```
for fa, fb in combinations(BIN_COMBOS, 2):
    print(f'For feature combo: {fa} + {fb}')
    extra_features = make_binary_combinations(fa, fb, df=train_df)
    unified_train_X = np.hstack((train_X, extra_features))

    extra_features = make_binary_combinations(fa, fb, df=eval_df)
    unified_test_X = np.hstack((eval_X, extra_features))

    apt = AvgPerceptron(epochs=5, n_features=unified_train_X.shape[1])
    apt.summary(trX=unified_train_X, trY=train_y, teX=unified_test_X,␣
 ↪teY=eval_y)
    print("\n\n")
```

```
For feature combo: age + sector
epoch 1 updates 1256 (25.12%) dev_error 16.10% (+:19.50%)
epoch 2 updates 1193 (23.86%) dev_error 15.00% (+:20.00%)
epoch 3 updates 1141 (22.82%) dev_error 15.10% (+:20.50%)
epoch 4 updates 1154 (23.08%) dev_error 15.50% (+:20.30%)
epoch 5 updates 1118 (22.36%) dev_error 15.80% (+:20.60%)



For feature combo: age + education
epoch 1 updates 1253 (25.06%) dev_error 15.30% (+:18.70%)
epoch 2 updates 1167 (23.34%) dev_error 15.30% (+:19.50%)
epoch 3 updates 1120 (22.40%) dev_error 15.90% (+:20.30%)
epoch 4 updates 1138 (22.76%) dev_error 15.80% (+:20.60%)
epoch 5 updates 1090 (21.80%) dev_error 15.50% (+:20.70%)



For feature combo: age + marital-status
epoch 1 updates 1228 (24.56%) dev_error 14.80% (+:19.00%)
epoch 2 updates 1178 (23.56%) dev_error 15.30% (+:19.90%)
epoch 3 updates 1170 (23.40%) dev_error 15.20% (+:19.80%)
epoch 4 updates 1180 (23.60%) dev_error 15.40% (+:19.80%)
epoch 5 updates 1167 (23.34%) dev_error 15.80% (+:20.00%)



For feature combo: age + occupation
epoch 1 updates 1252 (25.04%) dev_error 15.30% (+:19.30%)
epoch 2 updates 1162 (23.24%) dev_error 15.40% (+:20.00%)
epoch 3 updates 1149 (22.98%) dev_error 16.10% (+:20.70%)
epoch 4 updates 1086 (21.72%) dev_error 16.70% (+:20.90%)
epoch 5 updates 1095 (21.90%) dev_error 16.90% (+:20.90%)
```

```
For feature combo: age + race
epoch 1 updates 1256 (25.12%) dev_error 14.90% (+:19.90%)
epoch 2 updates 1208 (24.16%) dev_error 15.20% (+:20.20%)
epoch 3 updates 1169 (23.38%) dev_error 15.30% (+:20.10%)
epoch 4 updates 1160 (23.20%) dev_error 15.30% (+:20.70%)
epoch 5 updates 1156 (23.12%) dev_error 15.50% (+:20.50%)


For feature combo: age + gender
epoch 1 updates 1226 (24.52%) dev_error 15.60% (+:20.60%)
epoch 2 updates 1178 (23.56%) dev_error 15.20% (+:20.20%)
epoch 3 updates 1171 (23.42%) dev_error 15.50% (+:20.30%)
epoch 4 updates 1145 (22.90%) dev_error 15.30% (+:20.90%)
epoch 5 updates 1188 (23.76%) dev_error 15.60% (+:21.00%)


For feature combo: age + hours-per-week
epoch 1 updates 1260 (25.20%) dev_error 15.50% (+:18.50%)
epoch 2 updates 1165 (23.30%) dev_error 15.20% (+:19.80%)
epoch 3 updates 1160 (23.20%) dev_error 15.30% (+:20.10%)
epoch 4 updates 1122 (22.44%) dev_error 15.80% (+:20.20%)
epoch 5 updates 1090 (21.80%) dev_error 16.60% (+:19.80%)


For feature combo: age + country-of-origin
epoch 1 updates 1268 (25.36%) dev_error 15.10% (+:19.90%)
epoch 2 updates 1191 (23.82%) dev_error 14.80% (+:19.60%)
epoch 3 updates 1163 (23.26%) dev_error 15.10% (+:19.70%)
epoch 4 updates 1176 (23.52%) dev_error 14.60% (+:19.60%)
epoch 5 updates 1163 (23.26%) dev_error 14.80% (+:20.00%)


For feature combo: sector + education
epoch 1 updates 1234 (24.68%) dev_error 14.90% (+:18.90%)
epoch 2 updates 1201 (24.02%) dev_error 15.00% (+:19.60%)
epoch 3 updates 1206 (24.12%) dev_error 15.70% (+:19.70%)
epoch 4 updates 1147 (22.94%) dev_error 16.10% (+:19.70%)
epoch 5 updates 1156 (23.12%) dev_error 15.80% (+:20.00%)


For feature combo: sector + marital-status
```

```
epoch 1 updates 1256 (25.12%) dev_error 15.00% (+:19.00%)
epoch 2 updates 1168 (23.36%) dev_error 14.60% (+:19.60%)
epoch 3 updates 1171 (23.42%) dev_error 14.80% (+:19.60%)
epoch 4 updates 1155 (23.10%) dev_error 14.70% (+:19.90%)
epoch 5 updates 1121 (22.42%) dev_error 15.10% (+:20.10%)


For feature combo: sector + occupation
epoch 1 updates 1235 (24.70%) dev_error 15.70% (+:19.30%)
epoch 2 updates 1203 (24.06%) dev_error 15.40% (+:20.00%)
epoch 3 updates 1185 (23.70%) dev_error 15.60% (+:20.00%)
epoch 4 updates 1187 (23.74%) dev_error 15.50% (+:20.30%)
epoch 5 updates 1161 (23.22%) dev_error 15.90% (+:20.30%)


For feature combo: sector + race
epoch 1 updates 1235 (24.70%) dev_error 15.00% (+:18.40%)
epoch 2 updates 1175 (23.50%) dev_error 14.90% (+:18.90%)
epoch 3 updates 1156 (23.12%) dev_error 15.00% (+:19.20%)
epoch 4 updates 1193 (23.86%) dev_error 15.00% (+:20.20%)
epoch 5 updates 1150 (23.00%) dev_error 15.10% (+:19.90%)


For feature combo: sector + gender
epoch 1 updates 1261 (25.22%) dev_error 14.90% (+:19.10%)
epoch 2 updates 1173 (23.46%) dev_error 15.10% (+:19.30%)
epoch 3 updates 1156 (23.12%) dev_error 14.90% (+:19.70%)
epoch 4 updates 1157 (23.14%) dev_error 15.20% (+:19.80%)
epoch 5 updates 1179 (23.58%) dev_error 14.90% (+:19.90%)


For feature combo: sector + hours-per-week
epoch 1 updates 1251 (25.02%) dev_error 15.50% (+:19.10%)
epoch 2 updates 1181 (23.62%) dev_error 15.70% (+:19.50%)
epoch 3 updates 1164 (23.28%) dev_error 15.60% (+:19.60%)
epoch 4 updates 1155 (23.10%) dev_error 15.30% (+:19.50%)
epoch 5 updates 1162 (23.24%) dev_error 15.50% (+:19.90%)


For feature combo: sector + country-of-origin
epoch 1 updates 1237 (24.74%) dev_error 15.20% (+:19.00%)
epoch 2 updates 1189 (23.78%) dev_error 14.80% (+:19.20%)
epoch 3 updates 1176 (23.52%) dev_error 14.90% (+:19.90%)
```

```
epoch 4 updates 1166 (23.32%) dev_error 14.90% (+:20.50%)
epoch 5 updates 1165 (23.30%) dev_error 14.80% (+:20.20%)


For feature combo: education + marital-status
epoch 1 updates 1242 (24.84%) dev_error 15.40% (+:20.00%)
epoch 2 updates 1187 (23.74%) dev_error 15.60% (+:20.00%)
epoch 3 updates 1178 (23.56%) dev_error 15.20% (+:20.40%)
epoch 4 updates 1143 (22.86%) dev_error 15.30% (+:20.50%)
epoch 5 updates 1148 (22.96%) dev_error 15.60% (+:20.60%)


For feature combo: education + occupation
epoch 1 updates 1256 (25.12%) dev_error 15.50% (+:18.30%)
epoch 2 updates 1218 (24.36%) dev_error 15.10% (+:19.50%)
epoch 3 updates 1148 (22.96%) dev_error 15.10% (+:19.50%)
epoch 4 updates 1155 (23.10%) dev_error 15.50% (+:19.70%)
epoch 5 updates 1157 (23.14%) dev_error 15.60% (+:20.20%)


For feature combo: education + race
epoch 1 updates 1236 (24.72%) dev_error 15.70% (+:19.50%)
epoch 2 updates 1190 (23.80%) dev_error 15.40% (+:19.80%)
epoch 3 updates 1184 (23.68%) dev_error 15.70% (+:20.30%)
epoch 4 updates 1180 (23.60%) dev_error 15.70% (+:20.30%)
epoch 5 updates 1144 (22.88%) dev_error 15.60% (+:20.40%)


For feature combo: education + gender
epoch 1 updates 1255 (25.10%) dev_error 14.70% (+:18.70%)
epoch 2 updates 1165 (23.30%) dev_error 14.40% (+:19.20%)
epoch 3 updates 1179 (23.58%) dev_error 14.90% (+:19.70%)
epoch 4 updates 1164 (23.28%) dev_error 14.80% (+:20.20%)
epoch 5 updates 1150 (23.00%) dev_error 15.20% (+:20.00%)


For feature combo: education + hours-per-week
epoch 1 updates 1264 (25.28%) dev_error 14.60% (+:18.20%)
epoch 2 updates 1188 (23.76%) dev_error 15.30% (+:19.30%)
epoch 3 updates 1165 (23.30%) dev_error 15.00% (+:19.20%)
epoch 4 updates 1143 (22.86%) dev_error 15.30% (+:20.10%)
epoch 5 updates 1145 (22.90%) dev_error 15.40% (+:20.00%)
```

```
For feature combo: education + country-of-origin
epoch 1 updates 1271 (25.42%) dev_error 14.50% (+:18.30%)
epoch 2 updates 1173 (23.46%) dev_error 14.90% (+:20.50%)
epoch 3 updates 1194 (23.88%) dev_error 15.10% (+:20.30%)
epoch 4 updates 1187 (23.74%) dev_error 15.10% (+:20.30%)
epoch 5 updates 1147 (22.94%) dev_error 15.70% (+:20.70%)



For feature combo: marital-status + occupation
epoch 1 updates 1254 (25.08%) dev_error 15.30% (+:19.10%)
epoch 2 updates 1184 (23.68%) dev_error 14.80% (+:20.20%)
epoch 3 updates 1189 (23.78%) dev_error 14.70% (+:20.30%)
epoch 4 updates 1182 (23.64%) dev_error 14.90% (+:20.70%)
epoch 5 updates 1188 (23.76%) dev_error 14.90% (+:20.50%)



For feature combo: marital-status + race
epoch 1 updates 1253 (25.06%) dev_error 15.40% (+:19.80%)
epoch 2 updates 1209 (24.18%) dev_error 14.80% (+:20.20%)
epoch 3 updates 1195 (23.90%) dev_error 15.00% (+:21.00%)
epoch 4 updates 1200 (24.00%) dev_error 15.00% (+:21.00%)
epoch 5 updates 1176 (23.52%) dev_error 15.30% (+:21.10%)



For feature combo: marital-status + gender
epoch 1 updates 1226 (24.52%) dev_error 14.60% (+:18.80%)
epoch 2 updates 1186 (23.72%) dev_error 15.10% (+:20.10%)
epoch 3 updates 1177 (23.54%) dev_error 14.40% (+:20.00%)
epoch 4 updates 1184 (23.68%) dev_error 14.80% (+:20.60%)
epoch 5 updates 1160 (23.20%) dev_error 15.00% (+:20.60%)



For feature combo: marital-status + hours-per-week
epoch 1 updates 1251 (25.02%) dev_error 14.10% (+:19.10%)
epoch 2 updates 1208 (24.16%) dev_error 14.20% (+:19.60%)
epoch 3 updates 1173 (23.46%) dev_error 14.20% (+:19.60%)
epoch 4 updates 1150 (23.00%) dev_error 14.60% (+:19.60%)
epoch 5 updates 1159 (23.18%) dev_error 15.00% (+:20.00%)



For feature combo: marital-status + country-of-origin
```

```
epoch 1 updates 1258 (25.16%) dev_error 15.20% (+:19.00%)
epoch 2 updates 1192 (23.84%) dev_error 15.00% (+:19.20%)
epoch 3 updates 1182 (23.64%) dev_error 14.90% (+:19.70%)
epoch 4 updates 1194 (23.88%) dev_error 15.00% (+:20.00%)
epoch 5 updates 1148 (22.96%) dev_error 15.00% (+:20.40%)



For feature combo: occupation + race
epoch 1 updates 1262 (25.24%) dev_error 14.90% (+:18.10%)
epoch 2 updates 1199 (23.98%) dev_error 15.00% (+:20.00%)
epoch 3 updates 1178 (23.56%) dev_error 15.40% (+:20.60%)
epoch 4 updates 1172 (23.44%) dev_error 15.60% (+:20.20%)
epoch 5 updates 1169 (23.38%) dev_error 15.60% (+:20.60%)



For feature combo: occupation + gender
epoch 1 updates 1254 (25.08%) dev_error 14.10% (+:18.90%)
epoch 2 updates 1201 (24.02%) dev_error 14.60% (+:19.80%)
epoch 3 updates 1192 (23.84%) dev_error 14.40% (+:20.60%)
epoch 4 updates 1141 (22.82%) dev_error 14.50% (+:20.30%)
epoch 5 updates 1199 (23.98%) dev_error 14.60% (+:20.40%)



For feature combo: occupation + hours-per-week
epoch 1 updates 1248 (24.96%) dev_error 14.90% (+:20.70%)
epoch 2 updates 1186 (23.72%) dev_error 15.00% (+:21.00%)
epoch 3 updates 1152 (23.04%) dev_error 15.20% (+:20.60%)
epoch 4 updates 1152 (23.04%) dev_error 15.50% (+:20.70%)
epoch 5 updates 1128 (22.56%) dev_error 15.60% (+:21.00%)



For feature combo: occupation + country-of-origin
epoch 1 updates 1272 (25.44%) dev_error 14.80% (+:18.60%)
epoch 2 updates 1208 (24.16%) dev_error 14.00% (+:19.80%)
epoch 3 updates 1176 (23.52%) dev_error 14.20% (+:19.80%)
epoch 4 updates 1163 (23.26%) dev_error 14.70% (+:20.30%)
epoch 5 updates 1154 (23.08%) dev_error 15.10% (+:20.70%)



For feature combo: race + gender
epoch 1 updates 1277 (25.54%) dev_error 14.30% (+:18.70%)
epoch 2 updates 1203 (24.06%) dev_error 14.60% (+:19.40%)
epoch 3 updates 1189 (23.78%) dev_error 15.10% (+:20.10%)
```

```
epoch 4 updates 1181 (23.62%) dev_error 15.00% (+:20.40%)
epoch 5 updates 1170 (23.40%) dev_error 15.00% (+:20.60%)



For feature combo: race + hours-per-week
epoch 1 updates 1268 (25.36%) dev_error 15.40% (+:18.40%)
epoch 2 updates 1190 (23.80%) dev_error 15.10% (+:19.70%)
epoch 3 updates 1146 (22.92%) dev_error 15.50% (+:19.90%)
epoch 4 updates 1162 (23.24%) dev_error 15.50% (+:20.30%)
epoch 5 updates 1155 (23.10%) dev_error 14.80% (+:20.20%)



For feature combo: race + country-of-origin
epoch 1 updates 1247 (24.94%) dev_error 15.00% (+:19.00%)
epoch 2 updates 1182 (23.64%) dev_error 14.40% (+:19.60%)
epoch 3 updates 1161 (23.22%) dev_error 14.90% (+:20.50%)
epoch 4 updates 1191 (23.82%) dev_error 14.80% (+:20.40%)
epoch 5 updates 1166 (23.32%) dev_error 15.30% (+:20.50%)



For feature combo: gender + hours-per-week
epoch 1 updates 1263 (25.26%) dev_error 15.30% (+:18.90%)
epoch 2 updates 1179 (23.58%) dev_error 14.60% (+:19.80%)
epoch 3 updates 1161 (23.22%) dev_error 14.80% (+:20.20%)
epoch 4 updates 1174 (23.48%) dev_error 14.80% (+:20.60%)
epoch 5 updates 1157 (23.14%) dev_error 15.20% (+:20.20%)



For feature combo: gender + country-of-origin
epoch 1 updates 1252 (25.04%) dev_error 15.40% (+:19.40%)
epoch 2 updates 1191 (23.82%) dev_error 15.10% (+:20.10%)
epoch 3 updates 1187 (23.74%) dev_error 15.00% (+:20.60%)
epoch 4 updates 1186 (23.72%) dev_error 15.30% (+:20.50%)
epoch 5 updates 1165 (23.30%) dev_error 15.40% (+:20.60%)



For feature combo: hours-per-week + country-of-origin
epoch 1 updates 1243 (24.86%) dev_error 15.90% (+:18.90%)
epoch 2 updates 1195 (23.90%) dev_error 15.50% (+:19.90%)
epoch 3 updates 1185 (23.70%) dev_error 14.90% (+:19.30%)
epoch 4 updates 1165 (23.30%) dev_error 14.80% (+:19.80%)
epoch 5 updates 1162 (23.24%) dev_error 14.90% (+:19.50%)
```

[237]:
```
edu_marr_vector = make_binary_combinations("education", "marital-status",␣
↪df=train_df)
occ_gend_vector = make_binary_combinations("occupation", "gender", df=train_df)
marr_hours_vector = make_binary_combinations("marital-status",␣
↪"hours-per-week", df=train_df)
unified_train_X = np.hstack((train_X, edu_marr_vector, occ_gend_vector,␣
↪marr_hours_vector))

edu_marr_vector = make_binary_combinations("education", "marital-status",␣
↪df=eval_df)
occ_gend_vector = make_binary_combinations("occupation", "gender", df=eval_df)
marr_hours_vector = make_binary_combinations("marital-status",␣
↪"hours-per-week", df=eval_df)
unified_test_X = np.hstack((eval_X, edu_marr_vector, occ_gend_vector,␣
↪marr_hours_vector))

apt = AvgPerceptron(epochs=5, n_features=unified_train_X.shape[1])
apt.summary(trX=unified_train_X, trY=train_y, teX=unified_test_X, teY=eval_y)
```

```
epoch 1 updates 1237 (24.74%) dev_error 14.60% (+:20.40%)
epoch 2 updates 1152 (23.04%) dev_error 14.60% (+:20.60%)
epoch 3 updates 1167 (23.34%) dev_error 14.60% (+:21.20%)
epoch 4 updates 1136 (22.72%) dev_error 14.80% (+:21.20%)
epoch 5 updates 1129 (22.58%) dev_error 15.30% (+:21.90%)
```

Marital status and hours per week combination gave best error rate

0.4.3 Best model stats

[252]:
```
fa = "marital-status"
fb = "hours-per-week"
print(f'For feature combo: {fa} + {fb}')
extra_features = make_binary_combinations(fa, fb, df=train_df)
unified_train_X = np.hstack((train_X, extra_features))

extra_features = make_binary_combinations(fa, fb, df=eval_df)
unified_test_X = np.hstack((eval_X, extra_features))

apt = AvgPerceptron(epochs=1, n_features=unified_train_X.shape[1])
apt.summary(trX=unified_train_X, trY=train_y, teX=unified_test_X, teY=eval_y)
print("\n\n")
```

```
For feature combo: marital-status + hours-per-week
epoch 1 updates 1251 (25.02%) dev_error 14.10% (+:19.10%)
```

[290]:
```
extra_features = make_binary_combinations(fa, fb, df=test_df)
unified_test_X = np.hstack((np.ones((len(test_X), 1)), test_X, extra_features))

effective_W = apt.c * apt.W - apt.av_W
predictions = (np.dot(unified_test_X, effective_W) > 0)
# predictions[predictions == True] = ">50K"
# predictions[predictions == False] = "<=50K"

predictions_labels = []
for pred in predictions:
    if pred == True:
        predictions_labels.append(">50K")
    else:
        predictions_labels.append("<=50K")

with open("predictions.y.out", "w+") as out:
    out.write("\n".join(predictions_labels))

print(f'Positive rate on test set: {np.mean(predictions) * 100}')
```

```
Failed computing binary combinations for 2 examples.
Positive rate on test set: 19.8
```

- Best error rate on dev is 14.10%. Achieved when a couple of binary features are combinedly added.
- Postive rate is 19.10% compared to nearly 25% on train set.
- Positive rate on test set: 19.80%

0.5 Debriefing

1. Approximately how many hours did you spend on this assignment?
   - 7 hours
2. Would you rate it as easy, moderate, or difficult?
   - Moderate
3. Did you work on it mostly alone, or mostly with other people?
   - Mostly alone
4. How deeply do you feel you understand the material it covers (0%–100%)?
   - 80%
5. Any other comments?