Contribution Report

Discussion and Solve Problems for Questions 1, 2, 3, and 4:
Equal Contribution by both partners

Solution Write-up for Questions 1, 2, 3, and 4:
David Wu

Proof-reading for Questions 1, 2, 3, and 4:
Shu Wang

1. a. The code for `process_data` is included below.

```
def process_data(data, labels):
    """
    Preprocess a dataset of strings into vector representations.

    Parameters
    ----------
        data: numpy array
            An array of N strings.
        labels: numpy array
            An array of N integer labels.

    Returns
    -------
    train_X: numpy array
        Array with shape (N, D) of N inputs.
    train_Y:
        Array with shape (N,) of N labels.
    val_X:
        Array with shape (M, D) of M inputs.
    val_Y:
        Array with shape (M,) of M labels.
    test_X:
      Array with shape (M, D) of M inputs.
  test_Y:
      Array with shape (M,) of M labels.
  """

  # Split the dataset of string into train, validation, and test
  # Use a 70/15/15 split
  # train_test_split shuffles the data before splitting it
  # Stratify keeps the proportion of labels the same in each split

  # -- WRITE THE SPLITTING CODE HERE --

  train_X, temp_data, train_Y, temp_labels = train_test_split(data, labels, train_size = 0.7,
                                                              stratify = labels)

  val_X, test_X, val_Y, test_Y = train_test_split(temp_data, temp_labels, train_size = 0.5,
                                                  stratify = temp_labels)


  # Preprocess each dataset of strings into a dataset of feature vectors
  # using the CountVectorizer function.
  # Note, fit the Vectorizer using the training set only, and then
  # transform the validation and test sets.

  # -- WRITE THE PROCESSING CODE HERE --
  cv = CountVectorizer()
  train_X = cv.fit_transform(train_X)
  val_X = cv.transform(val_X)
  test_X  = cv.transform(test_X)

  # Return the training, validation, and test set inputs and labels

  # -- RETURN THE ARRAYS HERE --
  return train_X, train_Y, val_X, val_Y, test_X, test_Y
```

b. The code for `select_knn_model` is included below:

```
def select_knn_model(train_X, val_X, train_Y, val_Y):
    """
    Test k in {1, ..., 20} and return the a k-NN model
    fitted to the training set with the best validation loss.

    Parameters
    ----------
        train_X: numpy array
            Array with shape (N, D) of N inputs.
        train_X: numpy array
            Array with shape (M, D) of M inputs.
        train_Y: numpy array
            Array with shape (N,) of N labels.
        val_Y: numpy array
            Array with shape (M,) of M labels.

    Returns
    -------
    best_model : KNeighborsClassifier
        The best k-NN classifier fit on the training data
        and selected according to validation loss.
    best_k : int
        The best k value according to validation loss.
    """
    for k in range(1,21):
        neigh = KNeighborsClassifier(n_neighbors = k)
        neigh.fit(train_X, train_Y)

        # According to the handout, we choose the best k according to validation accuracy
        accuracy = neigh.score(val_X, val_Y)

        if k == 1:
            best_k = 1
            best_k_accuracy = accuracy
            best_model = neigh
        if accuracy > best_k_accuracy:
            best_k = k
            best_k_accuracy = accuracy
            best_model = neigh

        # According to the docstring, we select k according to validation loss.
        # The results do not change, but in case you explicitly wanted an algorithm
        # with 0-1 loss, it is included below.
        # Otherwise, please disregard the commented section.
        """
        loss = 0
        prediction = neigh.predict(val_X)
        for i in range(len(prediction)):
            if prediction[i] != val_Y[i]:
                loss += 1

        if k == 1:
            best_k = 1
            best_k_loss = loss
            best_model = neigh

        if loss < best_k_loss:
            best_k = k
            best_k_loss = loss
            best_model = neigh"""


    return best_model, best_k
```

The output from running the python code is:

```
Selected K: 9
Test Acc: 0.6591836734693878
```

This means that K = 9 has the best validation accuracy, and thus was selected. The test accuracy is around 0.659.

c. After changing `neigh = KNeighborsClassifier(n_neighbors = k)` to `neigh = KNeighborsClassifier(n_neighbors = k, metric = 'cosine')`, we indeed get an increase in test accuracy.

`metric = 'cosine'` uses $1 - \frac{\langle x,y \rangle}{\|x\| \, \|y\|}$ to calculate the distance between vectors *x* and *y*.

Consider `['cat', 'bulldozer', 'cat cat cat']` as given in the question. Using `CountVectorizer` gives a vocabulary of `['bulldozer', cat']` and the words are represented as `[0, 1], [1, 0], [0, 3]`, respectively.

Under Euclidean distance:

$d_{\text{Euclidean}}(cat, bulldozer) = \sqrt{2}$ and $d_{\text{Euclidean}}(cat, \ cat\ cat\ cat) = 2$

Under Cosine distance:

$d_{\text{cosine}}(cat, bulldozer) = 1$ and $d_{\text{cosine}}(cat, \ cat\ cat\ cat) = 1 - \frac{\sqrt{3}}{3} \approx 0.4226$

In K Nearest Neighbours, we want inputs that have the same label to be close to one another and inputs with different labels to be far away. This allow us to more easily draw a decision boundary.

News headlines may contain certain words that determine if it is real or fake, and headlines with questionable words appearing multiple times probably have the same label. As a result, we want words like 'cat' and 'cat cat cat' to have a small distance. From the example above, Euclidean distance gives a larger result (relative to $d(cat, bulldozer)$) than cosine distance, which is why we see an increase in accuracy after switching to `metric = 'cosine'`.

2. a. Since $X_i \overset{d}{=} X$ and $Y_i \overset{d}{=} Y$, $Z_i \overset{d}{=} Z$ for all $i \in \{1 \dots d\}$. This means that $E(Z_i) = E(Z)$ and $Var(Z_i) = Var(Z)$.

$E(R) = E(Z_1 + Z_2 \dots + Z_d)$ (using linearity of Expectations)

$= E(Z_1) + E(Z_2) + \dots E(Z_d) = \sum_{i=1}^{d} E(Z_i) = d\, E(Z)$

$Var(R) = Var(Z_1 + Z_2 \dots + Z_d)$ (✱)

Since all X_i and Y_i are independently drawn, Z_i is independent from Z_j if $i \neq j$. Thus (✱) becomes:

$Var(R) = Var(Z_1) + Var(Z_2) + \dots Var(Z_d) = \sum_{i=1}^{d} Var(Z) = d\, Var(Z)$.

3. Denote $h_i(x)$ as h_i for convenience. All summations are from 1 to m.

$$L(\bar{h}, t) = \frac{1}{2}\left(\frac{1}{m}\sum h_i - t\right)^2 = \frac{1}{2}\left[\left(\frac{1}{m}\sum h_i\right)^2 - \frac{2}{m}\sum h_i t + t^2\right] \quad (1)$$

Now consider a discrete random variable, D, with m (finitely many) states and $P(D = h_i) = \frac{1}{m}$. Then $E(D) = \frac{1}{m}\sum h_i$ and from the hint, $\left[\frac{1}{m}\sum h_i\right]^2 = E(D)^2 \leq E(D^2) = \frac{1}{m}\sum h_i^2$. (2)

Substituting (2) into (1) gives:

$$L(\bar{h}, t) \leq \frac{1}{2}\left[\frac{1}{m}\sum h_i^2 - \frac{2}{m}\sum h_i t + t^2\right]$$

$$= \frac{1}{m}\cdot\frac{1}{2}\left[\sum\left(h_i^2 - 2h_i t + t^2\right)\right]$$

$$= \frac{1}{m}\cdot\sum\frac{1}{2}(h_i - t)^2 = \frac{1}{m}\sum L(h_i, t).$$

4. We use the following derivation for a, b, c. Summations indexed by t and x sum over $\{0, 1\}$. Note $p(x, t) = p(t|x)\, p(x)$

$$R[\gamma] = \sum_t \sum_x L_{0\text{-}1}(\gamma(x), t)\, p(x, t)$$

$$= \sum_t \sum_x I(\gamma(x) \neq t)\, p(t|x)\, p(x).$$

Since the summations are finite, we can switch order of summation:

$$R[\gamma] = \sum_x \sum_t I(\gamma(x) \neq t)\, p(t|x)\, p(x)$$

$$= p(0)\left[\sum_t I(\gamma(0) \neq t)\, p(t|0)\right] + p(1)\left[\sum_t I(\gamma(1) \neq t)\, p(t|1)\right] \quad (\star)$$

4. a. Substituting $p(t|x) = 1/2$ for $t \in \{0,1\}$, $x \in \{0,1\}$ into (48):

$$R[\gamma] = \frac{1}{2} p(0) \left[\sum_t I(\gamma(0) \neq t) \right] + \frac{1}{2} p(1) \left[\sum_t I(\gamma(1) \neq t) \right]$$

Since $\gamma(0)$ takes either 0 or 1, Regardless of the value of $\gamma(0)$:

$$\sum_t I(\gamma(0) \neq t) = I(\gamma(0) \neq 0) + I(\gamma(0) \neq 1)$$

$$= I(\gamma(0) = 1) + I(\gamma(0) = 0) = 1$$

Similarly the second summation is also equal to 1 regardless of $\gamma(1)$.

Thus, for all maps $\gamma: \{0,1\} \to \{0,1\}$:

$$R[\gamma] = \frac{1}{2} p(0) + \frac{1}{2} p(1) = \frac{1}{2} (p(0) + p(1)) = \frac{1}{2} \cdot 1 = \frac{1}{2}$$

b. Take y^* in question and $x \in \{0,1\}$. Since $\mathbb{I}[y(x) \neq \operatorname{argmax}_t p(t|x)] \max_t p(t|x) = 0$,

$R[y^*] = p(0) \min_t p(t|0) + p(1) \min_t p(t|1)$ (from formula ☆)

Consider arbitrary y. $y(x) \in \{0,1\}$, so $\mathbb{I}(y(x) \neq t) = 1$ for exactly 1 value of t, when $t \in \{0,1\}$. Thus (☆) becomes:

$R[y] = p(0)\, p(t_1|0) + p(1)\, p(t_2|0)$ for some $t_1, t_2 \in \{0,1\}$.

However $p(t_1|0) \geq \min_t p(t|0)$ and $p(t_2|1) \geq \min_t p(t|1)$, so

$R[y] \geq R[y^*]$.

For uniqueness, consider y s.t. $R(y) = R(y^*)$. Since $p(0|x) \neq p(1,x)$ for all x, $\min_t p(t,x) \neq \max_t p(t|x)$ —— (1)

$R[y] = R[y^*]$ implies $\mathbb{I}[y(x) \neq \operatorname{argmax}_t p(t|x)] \max_t p(t|x) = 0$, $x \in \{0,1\}$, so

$y(x) = \operatorname{argmax}_t p(t|x)$. Note argmax_t returns a number since (1)

c

t \ x	0	1
0	1/8	1/4
1	1/8	1/2

Consider the table for the distribution of (x, t).

Note: $p(0|0) = 1/2$ $\quad p(0|1) = 1/3$

$p(1|0) = 1/2$ $\quad p(1|1) = 2/3$

From (*), the first summation only depends on $y(0)$ and the second only on $y(1)$. To min $R(y)$, we can minimize each summation independently.

Since $1/3 < 2/3$, we want $I(y(1) \neq 1)\, p(1|1) = 0$, implying $y(1) = 1$.

Since $p(0|0) = p(1|0)$, $I(y(0) \neq 0)\, p(1, 0) = I(y(0) \neq 1)\, p(1, 0)$,

so $y(0) = 0$ and $y(1) = 1$, both yeild a minimum. Therefore there are

two optimal predictors $y(0) = 0$, $y(1) = 1$ and $y(0) = 1$, $y(1) = 1$.