<u>Results:</u>

- ✓ “Describe each baseline and classifier in your Findings document.”
  - <u>Dummy Classifier (strategy = ‘most_frequent’):</u>
    - Used “DummyClassifier” from scikit-learn. Randomly predicting the outcome from class probabilities of the dev set.
    - Parameters:
      - **Strategy:** Default strategy is “prior” but used “most_frequent” as given in the question which concentrates on the most frequent class label.
      - **Random State:** To generate consistent results when the algorithm is running multiple times, we should not have any shuffling of the data. Hence, I have set the random state as “100” and used the same random state when running the experiments multiple times.

  - <u>Dummy Classifier (strategy = ‘stratified’):</u>
    - Used “DummyClassifier” from scikit-learn. Randomly predicting the outcome from class probabilities of the dev set.
    - Parameters:
      - **Strategy:** Default strategy is “prior” but used “stratified” as given in the question which concentrates on the class probabilities for randomly predicting the outcome.
      - **Random State:** For the above-mentioned reason(previous classifiers), used random state as “200” for generating consistent results.

  - <u>Random Forest:</u>
    - Used “RandomForestClassifier” from scikit-learn. It is basically a set of decision trees from a randomly selected subset of the training set.
    - Parameters:
      - **N estimators:** It is the number of trees in the forest. I used the default value 100, didn’t change it.
      - **Criterion:** When the split happens, this parameter measures the quality of the split. I have used the default value “gini”. Although “entropy” and “gini” work internally in similar fashion for measuring the quality of the split, I understand that “gini” has better computing power than “entropy”. So, for this reason I have used the default value “gini” provided by the random forest classifier.
      - **Random State:** For the above-mentioned reason(previous classifiers), used random state as “300” for generating consistent results.

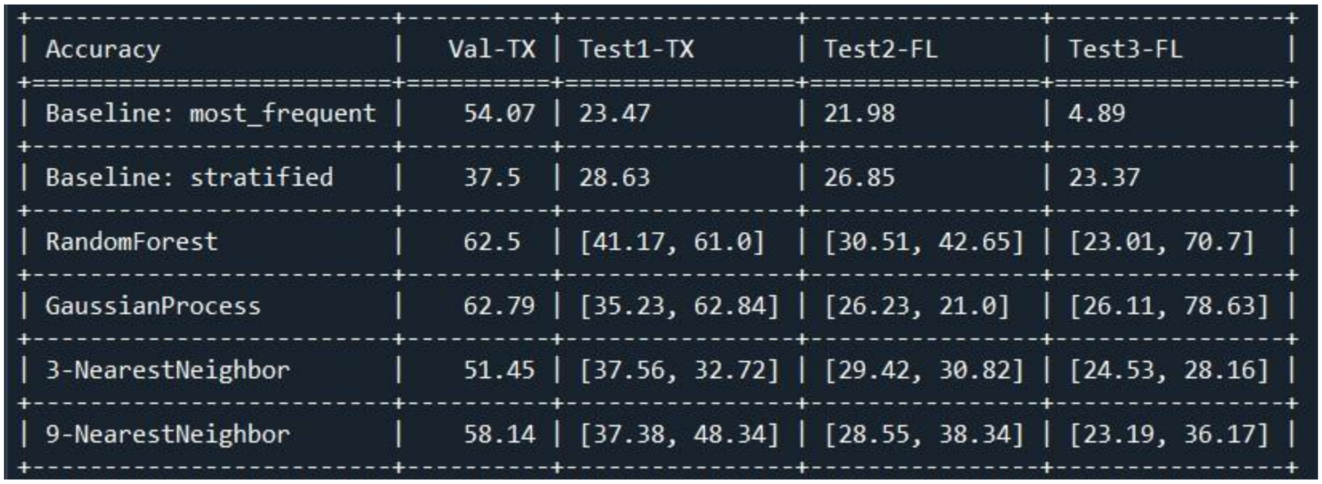
  - <u>Gaussian Process:</u>

    - Used “GaussianProcessClassifier” from scikit-learn. For approximating the non-Gaussian posterior by a Gaussian, the Laplace approximation is used.
    - Parameters:
        - **Kernel:** It is the covariance function of the gaussian process. I have used the kernel as “kernel = 1.0 * RBF(10.0)”
        - **Random State:** For the above-mentioned reason(previous classifiers), used random state as “400” for generating consistent results.

- <u>3-Nearest Neighbor:</u>
    - Used “KNeighborsClassifier” from scikit-learn. KNN algorithm assumes that similar things are close to each other. It captures the idea of similarity (proximity).
    - Parameters:
        - **N neighbors:** It is the number of neighbors to use for the kneighbors queries. It’s default value is 5, but in the question paper, it talks about 3-NN, where k = 3 which is the number of neighbors, so I have set the value of n neighbors as “3”.
        - **Weights:** With reference to scikit-learn documentation, I didn’t change this parameter, used the default value “uniform” to have equal weights for all the points in the neighborhood.

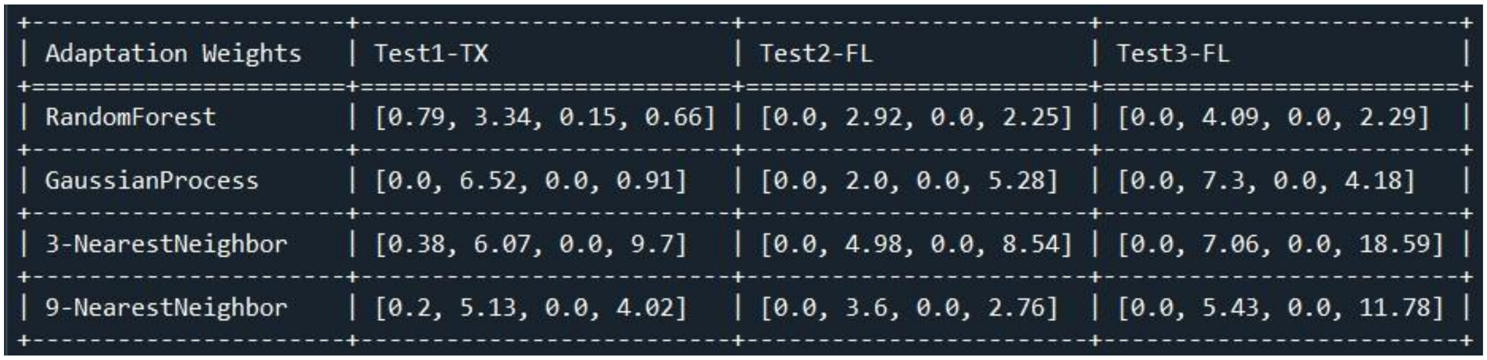
- <u>9-Nearest Neighbor:</u>
    - Used “KNeighborsClassifier” from scikit-learn. KNN algorithm assumes that similar things are close to each other. It captures the idea of similarity (proximity).
    - Parameters:
        - **N neighbors:** It is the number of neighbors to use for the kneighbors queries. It’s default value is 5, but in the question paper, it talks about 9-NN, where k = 9 which is the number of neighbors, so I have set the value of n neighbors as “9”.
        - **Weights:** With reference to scikit-learn documentation, I didn’t change this parameter, used the default value “uniform” to have equal weights for all the points in the neighborhood.

6. Report results in your Findings document:



```
+-------------------------+---------+----------------+----------------+----------------+
| Accuracy                |  Val-TX | Test1-TX       | Test2-FL       | Test3-FL       |
+=========================+=========+================+================+================+
| Baseline: most_frequent |   54.07 | 23.47          | 21.98          | 4.89           |
+-------------------------+---------+----------------+----------------+----------------+
| Baseline: stratified    |   37.5  | 28.63          | 26.85          | 23.37          |
+-------------------------+---------+----------------+----------------+----------------+
| RandomForest            |   62.5  | [41.17, 61.0]  | [30.51, 42.65] | [23.01, 70.7]  |
+-------------------------+---------+----------------+----------------+----------------+
| GaussianProcess         |   62.79 | [35.23, 62.84] | [26.23, 21.0]  | [26.11, 78.63] |
+-------------------------+---------+----------------+----------------+----------------+
| 3-NearestNeighbor       |   51.45 | [37.56, 32.72] | [29.42, 30.82] | [24.53, 28.16] |
+-------------------------+---------+----------------+----------------+----------------+
| 9-NearestNeighbor       |   58.14 | [37.38, 48.34] | [28.55, 38.34] | [23.19, 36.17] |
+-------------------------+---------+----------------+----------------+----------------+
```

7. Show the adaptation weights that were calculated for each classifier and each test set in your Findings document:



```
+--------------------+-------------------------+------------------------+-------------------------+
| Adaptation Weights | Test1-TX                | Test2-FL               | Test3-FL                |
+====================+=========================+========================+=========================+
| RandomForest       | [0.79, 3.34, 0.15, 0.66] | [0.0, 2.92, 0.0, 2.25] | [0.0, 4.09, 0.0, 2.29]  |
+--------------------+-------------------------+------------------------+-------------------------+
| GaussianProcess    | [0.0, 6.52, 0.0, 0.91]  | [0.0, 2.0, 0.0, 5.28]  | [0.0, 7.3, 0.0, 4.18]   |
+--------------------+-------------------------+------------------------+-------------------------+
| 3-NearestNeighbor  | [0.38, 6.07, 0.0, 9.7]  | [0.0, 4.98, 0.0, 8.54] | [0.0, 7.06, 0.0, 18.59] |
+--------------------+-------------------------+------------------------+-------------------------+
| 9-NearestNeighbor  | [0.2, 5.13, 0.0, 4.02]  | [0.0, 3.6, 0.0, 2.76]  | [0.0, 5.43, 0.0, 11.78] |
+--------------------+-------------------------+------------------------+-------------------------+
```

9. Answer these questions in your Findings document:

A. In your own words, explain "label shift" and give an example.

Answer:

/* Reference Citation: I have used my readings – R5 submission */

- ✓ It is a kind of domain shift where we can observe that the feature distribution remains same, but the output (labels) distribution may change. It makes anticausal assumption(predicting causes). It is the opposite of covariate shift.
- ✓ Example 1: We train a model to classify activity state (lying down, sitting) for elder people and then deploy same people while exercising.
- ✓ Example 2: Train on tv broadcasting data before election and then test on tv broadcasting data after election – some parameters will be changed as new people and discussions would have added to the before election situation.

B. In your own words, explain how BBSC works.

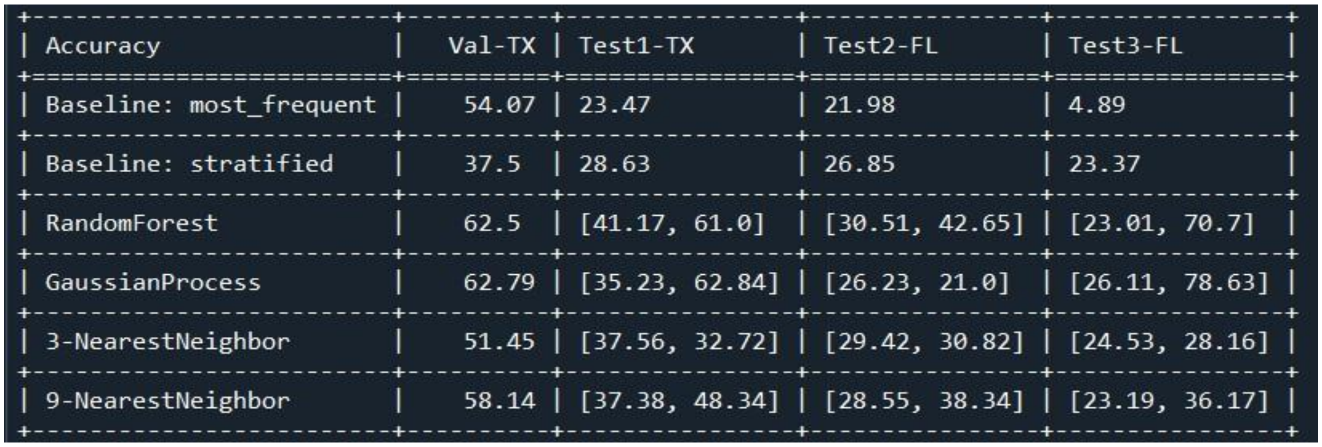
Answer:

- ✓ Black Box Shift Correction (BBSC) is a method for correcting label shift. In this approach we adjust the predictions based on classifier reliability and class prevalence.
- ✓ BBSC actually starts working only after BBSD, that is Black Box Shift Detection, once the shift is detected, this approach can be used to make the shift correction.
- ✓ First, we train and model and do a test on the validation test. Then we predict the results on a test set.
- ✓ Using the validation set true values and prediction values, we construct the confusion matrix. From this confusion matrix, we will try to understand which label prediction is actually less reliable or more reliable.
- ✓ Now using this reliability from the validation set confusion matrix, by checking the test set results prediction distribution, we will try to estimate the new class weights. This estimation is actually based on the classifier reliability (which we get from validation set confusion matrix) and class prevalence (prediction distribution from the test set).
- ✓ Once we estimate the new weights, using the test set predictions, probability we will calculate the new predictions.
- ✓ The estimation of weights mainly depends on the reliability as well as the prediction distribution in the test set. For example, when two classes are very likely in the validation data and the validation predicts with 75% reliability (for both classes) and test set has 50% each distribution in the prediction result. In this case, we can observe that reliability and prediction distribution give equal chance to both the classes. In such cases, the weights would be "1.0" for both the classes. In case, if for the same scenario of same reliability for both the classes, if the test set distribution is different let say (70% and 30%), then the weight of 70% class will be increased and same amount of weight will be decreased for 30% class.
- ✓ The estimated new weights will be used either for re-training the classifier or for modifying the posteriors.

C. In your experiments, in which cases did BBSC adaptation improve test set accuracy? How was better accuracy achieved?

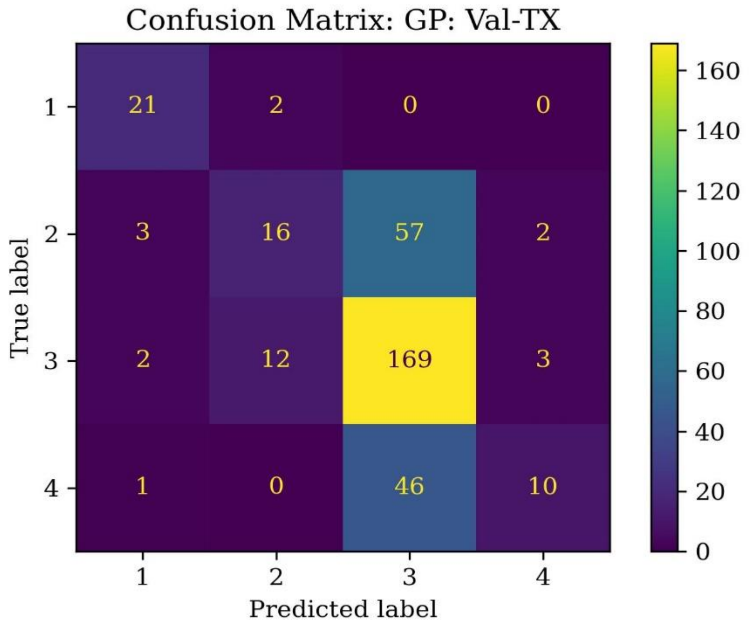
<u>Answer:</u>

- ✓ In the total combination of 4 classifiers and 3 test sets, I can see that for almost 10 combinations, the test set accuracy is improved after BBSC adaptation as shown in the below tabular form.



```
+-------------------------+---------+----------------+----------------+----------------+
| Accuracy                |  Val-TX | Test1-TX       | Test2-FL       | Test3-FL       |
+=========================+=========+================+================+================+
| Baseline: most_frequent |   54.07 | 23.47          | 21.98          | 4.89           |
+-------------------------+---------+----------------+----------------+----------------+
| Baseline: stratified    |   37.5  | 28.63          | 26.85          | 23.37          |
+-------------------------+---------+----------------+----------------+----------------+
| RandomForest            |   62.5  | [41.17, 61.0]  | [30.51, 42.65] | [23.01, 70.7]  |
+-------------------------+---------+----------------+----------------+----------------+
| GaussianProcess         |   62.79 | [35.23, 62.84] | [26.23, 21.0]  | [26.11, 78.63] |
+-------------------------+---------+----------------+----------------+----------------+
| 3-NearestNeighbor       |   51.45 | [37.56, 32.72] | [29.42, 30.82] | [24.53, 28.16] |
+-------------------------+---------+----------------+----------------+----------------+
| 9-NearestNeighbor       |   58.14 | [37.38, 48.34] | [28.55, 38.34] | [23.19, 36.17] |
+-------------------------+---------+----------------+----------------+----------------+
```

- ✓ We see increased accuracies in most of the cases after the BBSC adaptation as we can observe that the label shift issue is handled by estimating the weights correctly using the class reliability and the class prevalence. One example, is the random forest with test set "Test1-TX" where is an increase in the test accuracy for gaussian process classifier.
- ✓ Confusion matrix for Gaussian Process on validation set is as below.



- ✓ <u>Distribution of predictions on the test set, Test1-TX</u>
  Distribution of predictions of test1_tx: gaussian process is:
  [12574   16] for labels [2 4]
  Which implies the value is 0 for labels 1 and 3.
- ✓ <u>Adaptation Weights</u>
  [0.0, 6.52, 0.0, 0.91]
- ✓ From the confusion matrix, we can see that class 4 has good reliability and with test prediction distribution we can see that the prevalence of the class is also more compared to the class label 4. Hence, the weights are applied to recalculate the new predictions with the new estimated weights. This shows improvement in the test accuracy.
- ✓ Similarly, based on the combination of reliability and prevalence the weights are adjusted and the accuracy is improved after using the BBSC.

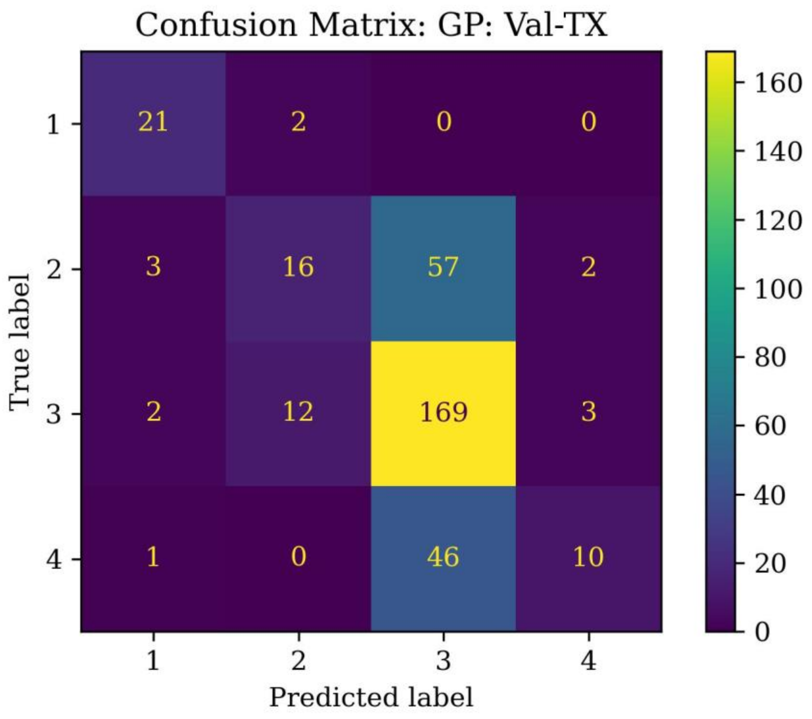
D. Did BBSC ever yield lower accuracy for a classifier after adaptation? If so, why do you think this happened? (Look at the classifier's validation set confusion matrix, its distribution of predictions on the test set where worse results were seen after adaptation, and the BBSC adaptation weights that were computed.)

Answer:

✓ Yes, in two cases as below I see that BBSC yielded lower accuracy for a classifier after adaptation. Although it reported lower accuracy, the margin between the actual accuracy and new accuracy is very low. (~ 5%)

- Classifier: Gaussian Process, Test Set: Test2-FL

  Confusion Matrix, Gaussian Process – validation dataset



Distribution of predictions on the test set, Test2-FL

Distribution of predictions of test2_fl: gaussian process is:

[335 3241] for labels [2 4]

Which implies, for labels 1 and 3, values are 0

Adaptation Weights

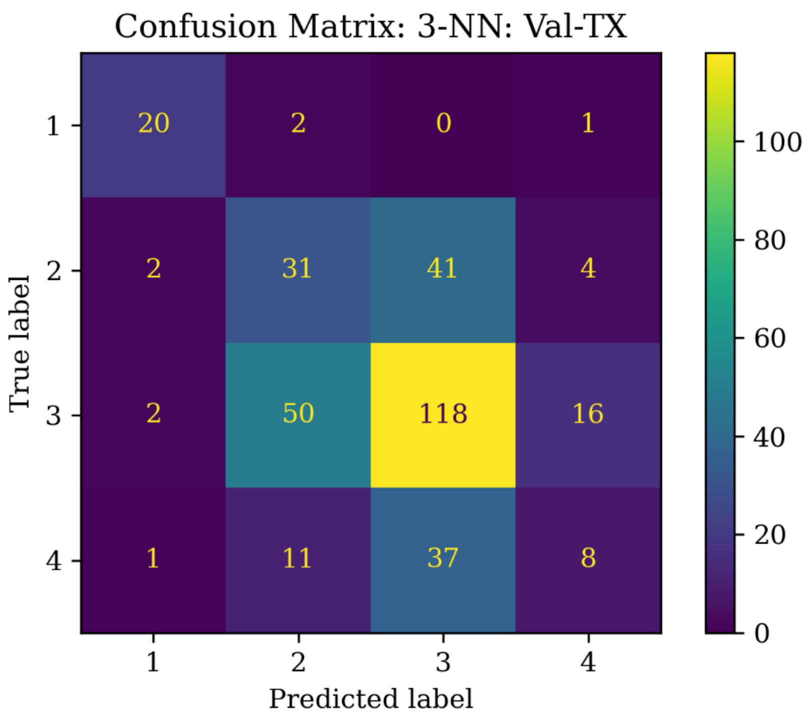
[0.0, 2.0, 0.0, 5.28]

The confusion matrix says the reliability is more with class label 1 but the prediction labels don't contain class label 1 or are present with a very low-class relevance. Hence, the weights of the labels present in the distribution set (2 and 4) are given some value and the other labels are given the weights as "0". Since BBSC focused more on the class relevance than the class reliability here, the weights are adjusted accordingly, and it resulted in a slight decrease of the test accuracy.

In similar fashion, the other scenario where the accuracy is less is as below.

- Classifier: 3-Nearest Neighbor, Test Set: Test1-TX

  Confusion Matrix, Gaussian Process – validation dataset



Distribution of predictions on the test set, Test1-TX

Distribution of predictions of test1_tx: 3-nn is:

[223 5223 7144] for labels [1 2 4]

Which implies, for label 3, value is 0.

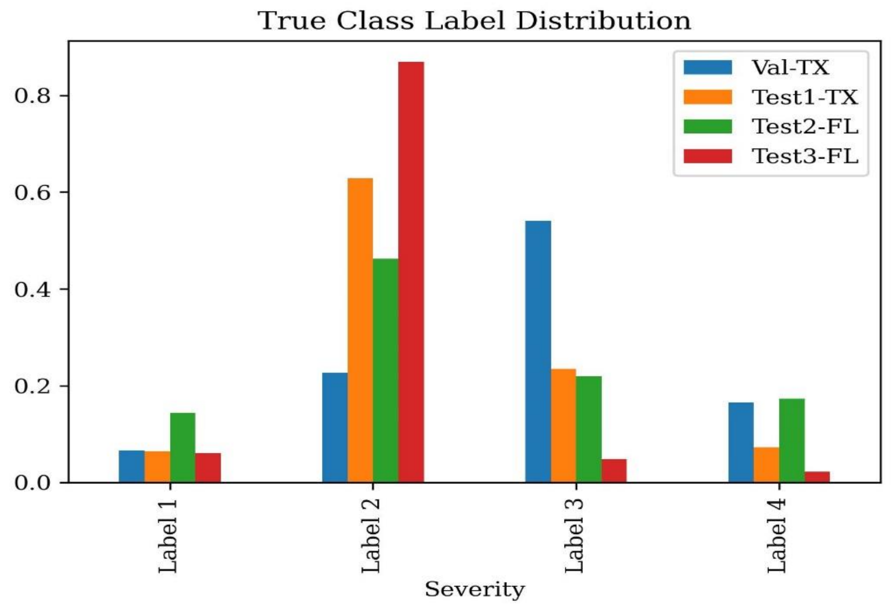
Adaptation Weights

[0.38, 6.07, 0.0, 9.7]

E. What do you learn by looking at the distribution of true class labels in the three test sets? How did car accident severity change in each data set (vs. validation)?

Answer:

✓ The distribution of true class labels in the validation set and the three test sets are as shown in the plot below.



- ✓ Overall, among the three test sets, we can observe that each dataset has more class distribution with severity 2 (label 2) whereas for validation dataset, it has more class distribution with severity 3.
- ✓ In the three test sets, we can see that the severity 1, has less distribution which means the car accidents are in general more severe (more the number → more the severity).
- ✓ In validation dataset, we can see the severity 3 has the highest distribution but it wasn't the case with the three test sets. This indicates that the data collected as part of the test in Texas during the period Jan 1 to Mar 1, 2020, reported the car accidents to be more severe (label 3).
- ✓ So, all in all, if this data is to be believed, we can say the severity of the car accidents is mostly less in Florida during Jan 1 to October 31, 2020, and in Texas during April 1 to October 31, 2020, when compared with the random subset data of Texas (Val-TX). This actually indicates the label shift.

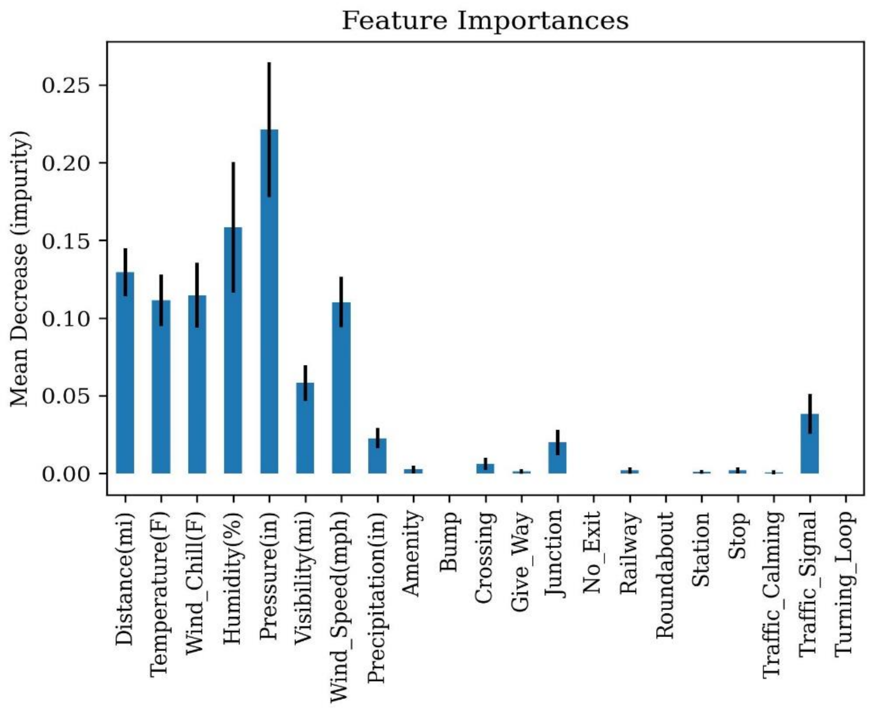
F. What is one reason that would cause all BBSC adaptation weights to be 0.0? Use your understanding of BBSC and inspection of the methods in label_shift_adaptation.py to answer this question.

<u>Answer:</u>

- ✓ I understand that below scenarios can cause the adaptation weights to be 0.0
  - In the test set, if any classes occur rarely in the set then such scenario will produce negative importance weights. In such cases, the weights will be clipped to 0.
  - If any class never appears in a test set, the weight will be 0.
  - If any of the classes never appears in the validation set true labels or pred labels, then we cannot invert the validation confusion matrix, in such case BBSC will send the required error message and will return the weights as 0.

10. Rank the features by their importance/relevance for the random forest classifier. (In scikit-learn, you can inspect clf.feature_importances_ after training a RandomForestClassifier in clf). Show the ranked list. Are the most important features what you would expect? Are there any surprises?

Answer:



- ✓ Above plot shows the most important features when I used the classifier random forest. I used the scikit-learn library feature importance based on mean decrease in impurity.
- ✓ I see that feature like “Pressure”, “Humidity”, “Distance”, and “Wind Speed” are the most important features compared to other features.
- ✓ To me, there is not much surprise in the features list.