Naive Bayes Classifier from scratch

תוכן

פרק 1: כללי

הסבר כללי על הפרויקט

הפרויקט נועד לבנות מסווג מסוג Naive Bayes מאפס , כולל כל השלבים המקדימים.

התוכנית מקבלת נתיב מהמשתמש אשר יש בו שלושה קבצים:

Structure.txt- קובץ אשר מציין את מבנה ה-dataset , כלל המאפיינים הקיימים בו וכל הערכים התקינים עבור כל מאפיין ומאפיין.

train.csv-קובץ אשר בתוכו יש את ה-dataset שעל בסיסו התוכנית אמורה לבנות את הסטטיסטיקה.

תחילה יופעל תהליך של pre-processing שבו תתבצע השלמה של ערכים חסרים או לחילופין ערכי רעש , כאלו שלא אמורים להיות שם.

- עבור ערכים נומריים נשלים בעזרת הערך הממוצע של המאפיין.
- עבור ערכים קטגוריאליים נשלים בעזרת הערך השכיח של המאפיין.

בנוסף, עבור ערכים נומריים תתבצע חלוקה לבינים(bins) לפי כמות בינים אשר המשתמש יכניס ועל פי עקרון ה-equal width binning .

לאחר שתהליך ה pre-processing מסתיים התוכנית עוברת לשלב חישוב ההסתברות המותנית, (ערך של מאפיין בהינתן הclass), לצורך הפשטות ובכדי למנוע הסתברות 0 השימוש הוא ב-m-estimator 1(Laplacian estimator).

test.csv- קובץ שיש בתוכו dataset שאותו עלינו לסווג על בסיס הנתונים הסטטיסטים אשר חישבנו מהקובץ הקודם.

בסופו של דבר, התוכנית תיצור קובץ חדש בשם output.txt שאליו היא תרשום את הסיווג של כל שורה ושורה.

קווים מנחים

הפרויקט נבנה בצורה אשר נותנת חשיבות לתכנות מונחה עצמים, כימוס והסתרת מידע, נוחות שימוש ,אי-תלות במערכת הפעלה, ממשק גרפי ,תקשורת בין הממשק הגרפי אל שאר התוכנית ע"י תבנית מסוג Observer (למטרת הצגת התקדמות החישובים למשתמש), ומדולריות(תבנית MVC).

פרק 2: מבנה התוכנית

התוכנית בנויה מ-5 קבצים שונים כאשר לצורך הפשטות, חילקנו את הפרויקט לארבעה מחלקות שונות.

- Main.py- הקובץ אחראי להפעיל את הממשק גרפי.
- Controller.py-מחלקה אשר אחראית על קישור בין הממשק הגרפי לבין החלק הלוגי של התוכנית.
- view.py-מחלקה אשר אחראית על כל הממשק גרפי של התוכנית.
- BuildClasifyer.py-מחלקה אשר אחראית על תהליך הpreprocessing ובניית הסטטיסטיקות על בסיס קובץ ה-Train.
- Classify.py-מחלקה אשר אחראית על סיווג קובץ הTest.

הערות כלליות הנוגעות לכלל הקבצים והמחלקות:

- בכדי למנוע אי תאימות למערכת הפעלה מסוימת(כמו למשל לכיוון הסלאש במערכות הפעלה שונות), מתבצע שימוש במודול OS בכל הנוגע לעבודה מול הנתיב ו/או שמות הקבצים.
- בכל התעסקות עם קבצים הקובץ נפתח על ידי הפקודה with open במקום ב- open בלבד . וזאת בכדי שהניהול קובץ התנהל אוטומטית(סגירת קובץ וכו').
- במחלקות BuildClasifyer ו-Classify כל השדות מוגדרים כPrivate בכדי למנוע שינוי לא רצוי של הנתונים.
- כל הקריאות לפונקציות מנוהלות מהבנאי וזאת לצורך עבודה אוטומטית של כלל התהליכים.
- משיקולים של זמן ריצה, ברוב המקומות שהיה צריך לחשב כמות חזרות, העדפנו להשתמש ב-Counter המובנה של השפה במקום כל מיני קומבינציות של map ו-count .
- בשל שיקולי זמן ריצה על פני שיקולי סיבוכיות זיכרון , אין שימוש בשיטת ה-lazy.
- בכל מקום אשר ישנו צורך בבדיקה אם ערך מסוים הוא מספרי נעשה שימוש בפונקצית בדיקה אשר לא מובנת בשפה(שכן, הפונקציה הרלוונטית קיימת אך ורק בגרסא 3 ומעלה). הפונקציה:

```
def __is_int(self,string):
    """function to check if a given string is an int"""
    try:
        int(string)
        return True
    except ValueError:
        return False
```

בדומה לזאת קיימת גם פונקציה הבודקת float.

מספר הנחות:

- אין כל בעיה בעמודת הCLASS והיא קיימת באינדקס האחרון של הקובץ.
- סדר העמודות זהה בשתי ה- DATASETS.
- אין ערכים חסרים בקובץ הtest וזאת בגלל העובדה שכל שורה אמורה לעבור סיווג באופן עצמאי וללא כל תלות בשורות האחרות (דבר זה לא יהיה רלוונטי אם נבצע השלמת ערכים חסרים בקובץ זה).

Main.py

הקובץ אחראי על יצירת הדברים הבאים:

- root frame עבור ה-gui והפעלת ה-mainloop שלו.
- אובייקט מסוג controller אשר יהיה אחראי על כל התקשורת בין המודולים של התוכנית(הממשק הגרפי והלוגיקה).

Controller.py

הבנאי של המחלקה קורא לבנאי של מחלקת view וזאת בכדי לבנות ולהפעיל את הממשק הגרפי.
כמו כן, הקריאה לבנאי נשמרת ב reference וזאת בכדי שה- controller יוכל לשלוח הודעות ועדכונים ל-View.

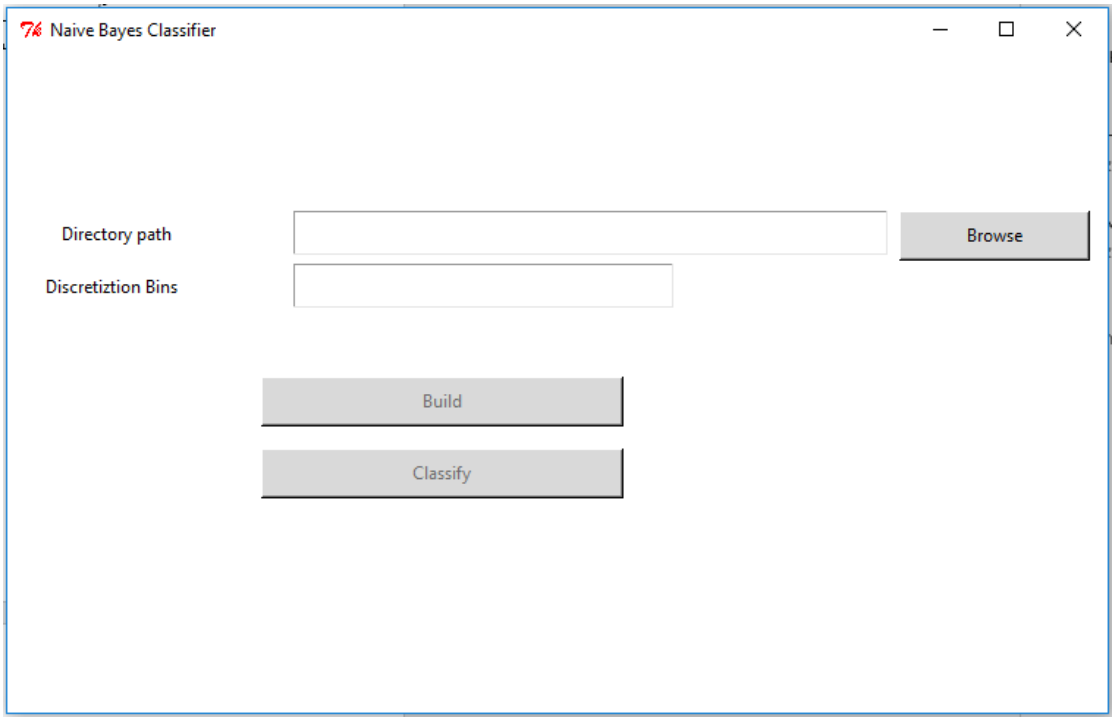
בנוסף, בcontroller יש את הפונקציונליות הבאה:

- בדיקת הנתיב אשר הוכנס.
- פונקציית עדכון אשר מקבלת עדכונים(הודעות ופרמטרים להעברה) מה-model(classify ו-BuildClasifyer) ומעבירה אותם אל ה-view(לצורך הצגתם למשתמש) או ל-class השני של ה-model.
- הפעלה של בנאיי ה-model .

View.py

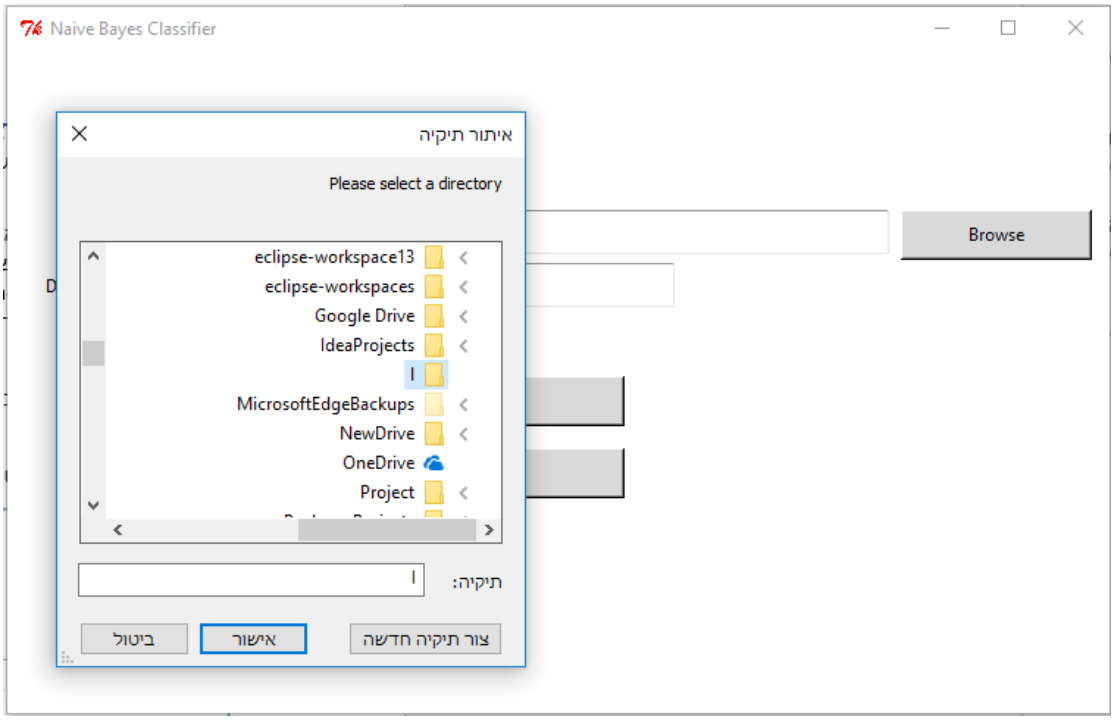
מחלקה אשר אחראית על בנייתו ותפעולו של ממשק המשתמש.

החלון הראשי של הממשק הוא:



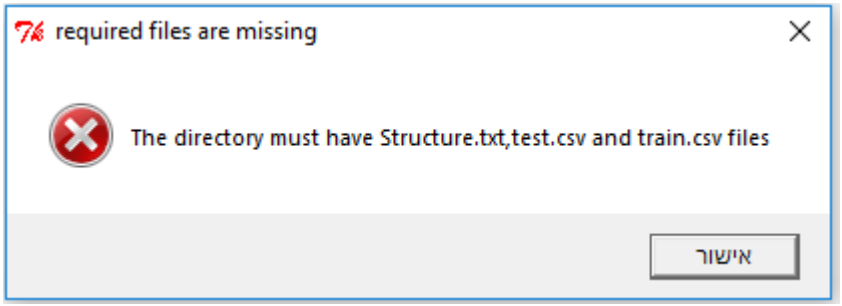
לצורך הפשטות, כל הממשק מנוהל ישירות מהבנאי בצורה אוטומטית.

כברירת מחדל, הדבר היחיד אשר נמצא במצב enabled הוא כפתור ה-Browse , בלחיצה על הכפתור הזה על המשתמש לבחור את הנתיב שבתוכו יש את שלושת הקבצים המצוינים מעלה.

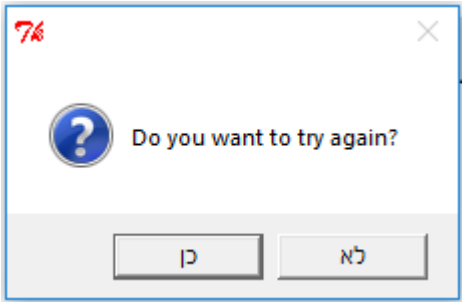


אם בחירת הנתיב על ידי המשתמש תופעל פונקציית בדיקת הנתיב(אשר נמצאת ב-controller), הפונקציה תבדוק אם אכן יש בתוך הנתיב את כל הקבצים הרצויים .

במידה ואינם קיימים, תופיע ההודעה הבאה:



ומיד אחריה שאלה אם המשתמש רוצה לנסות שוב או לא :

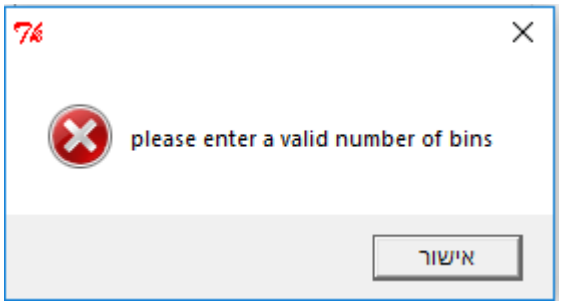


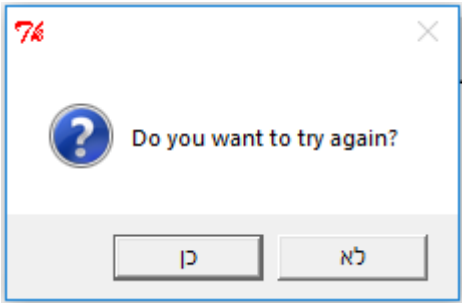
במידה והם כן קיימים כפתור ה-build וה- bin entry יעברו למצב מאופשר והמשתמש יוכל להמשיך בתהליך.

כעת, על המשתמש להכניס כמות bin-ים וללחוץ על build להמשך תהליך.

לחיצה על כפתור ה- build תפעיל את הפונקציה המתאימה ב-controller, תחילה תתבצע בדיקה אם כמות הבינים תקינה.

במידה והמשתמש לא הכניס כמות בינים או הכניס כמות בינים לא תקינה, תופיע הודעת שגיאה ומיד לאחריה הודעה אשר שואלת את המשתמש אם הוא מעוניין לנסות שוב:

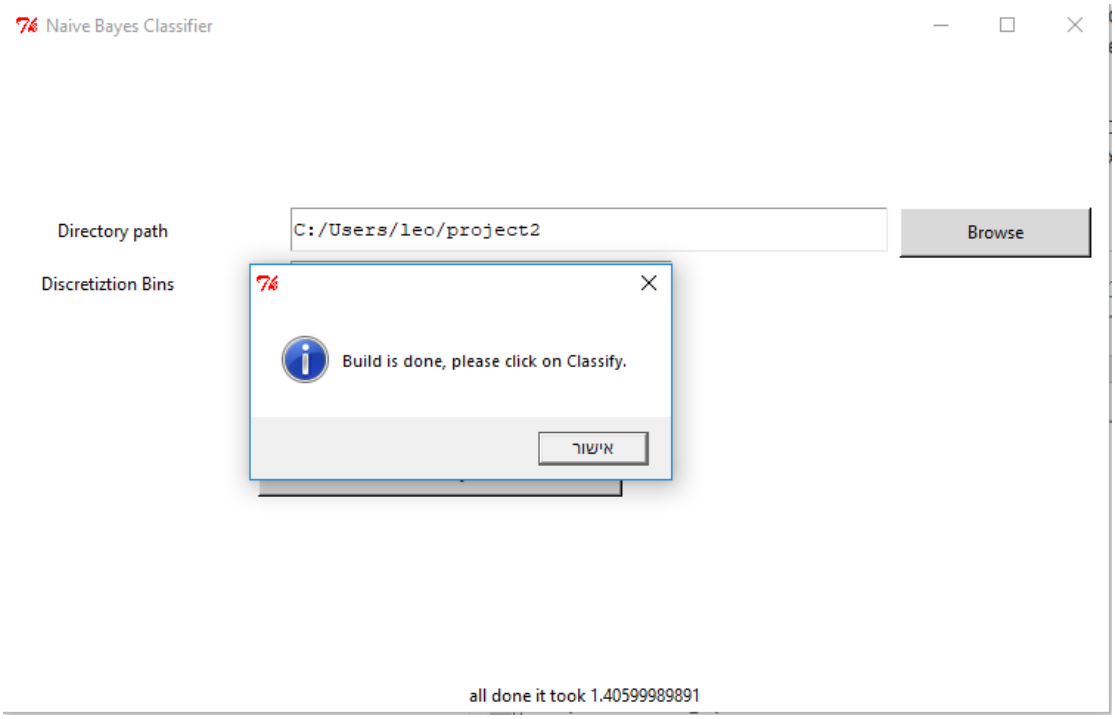




במידה והכל תקין, התוכנית תיצור אובייקט מסוג מחלקת BuildClasifyer ותעביר אליו את הנתיב של הקבצים את כמות הבינים הרצויה ו-reference לcontroller(אשר ישמש כObserver עבור האובייקט הנוצר).

הערה חשובה: בכדי למנוע לחיצה כפולה על ה-BUILD בשלב זה התוכנית מכבה את הכפתור.

ה-BuildClasifyer יבצע תהליך של pre-processing וחישוב סטטיסטי על קובץ ה-train , ולאחר סיום העבודה של ה- BuildClasifyer תוצג ההודעה הבאה:

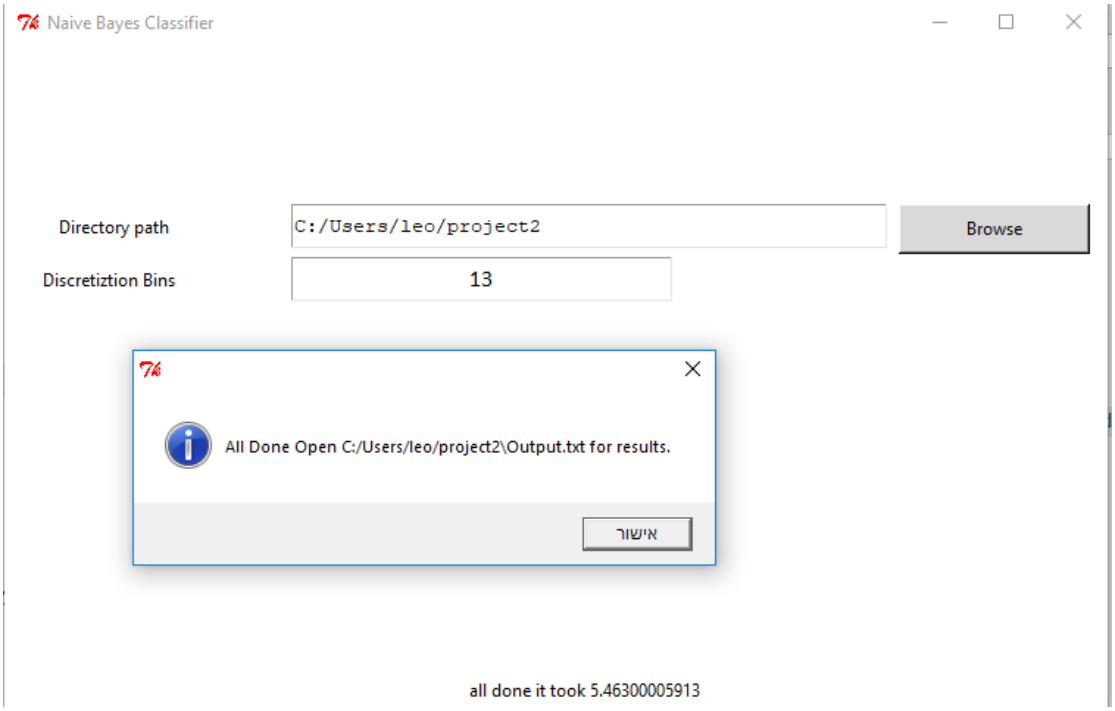


מאחורי הקלעים של התוכנית, ה-BuildClasifyer מודיע ל controller על סוף פעילות ומעביר אליו את כל הארגומנטים אשר יהיו רלוונטיים לעבודתו של ה-Classify :

- מילון אשר מחזיק את הטווחים של הבינים(וזאת בכדי לבצע חלוקה של הערכים הנומריים ב-test לבינים.
- רשימה המחזיקה את כל האינדקסים של הערכים הנומריים(וזאת בכדי שנדע על איזה ערכים לבצע את החלוקה לבינים).
- מילון הסטטיסטיקות.
- מילון ה-K(במידה ויהיה צורך לחשב סטטיסטיקות לערכים אשר לא הופיעו ב-train).
- מילון אשר מחזיק את כמות החזרות של כל אחד מה- class-ים(מאותה סיבה כמו מעלה).

ברגע שהמשתמש ילחץ על כפתור ה- Classify יופעל השלב האחרון של התוכנית, ה controller יעביר את כל הארגומנטים הרלוונטיים לבנאי של ה -Classify , כולל הנתיב לקובץ ה- test ו-reference ל-controller.

ה-Classify יבצע את תהליך הסיווג של כל שורה ושורה בקובץ ה-test ובסיום התהליך תוצג הודעה המודיעה על הצלחה עם קישור לקובץ התוצאות:


Naive Bayes Classifier
Directory path
C:/Users/leo/project2
Browse
Discretiztion Bins
13
All Done Open C:/Users/leo/project2\Output.txt for results.
אישור
all done it took 5.46300005913

BuildClasifyer

תהליך העבודה של ה-BuildClasifyer

1. קריאת קובץ ה-train.
2. ביצוע transpose לשם נוחות העבודה.
3. עבודה על ערכים נומריים:
   3.1. המרה (לאחר ביצוע בדיקה אם זה אפשרי) ממחרוזת לערך float.
   3.2. השלמה של חסרים או/ו של רעש, ההשלמה מתבצעת על ידי הכנסת ערך הממוצע.
   3.3. חלוקה ל-bin-ים על-פי עקרון ה- equal-width binning ע"י הנוסחא הבאה: $\left\lceil \frac{maxval - minval}{num\ of\ bins} \right\rceil$ למען הנוחות, אנו מחליפים את הערך בגבול המקסימלי של ה-bin.
   3.4. חישוב סטטיסטיקות על פי הנוסחא.
4. עבודה על ערכים קטגוריאליים:
   4.1. השלמה של חסרים או/ו של רעש, ההשלמה מתבצעת על ידי הכנסת הערך השכיח.
   4.2. חישוב סטטיסטיקות.

הערה: החישוב סטטיסטי הוא על פי הנוסחא הבאה: $\frac{n_c + 1}{|n| + k}$ (Laplacian estimator), כאשר:

n_c-כמות החזרות של ערך מסוים בהינתן class מסוים.

$|n|$-כמות החזרות של ה-class.

K - כמות ה"אופציות" של כל attribute.

פונקציות עיקריות ב- BuildClasifyer

- read_structure: הפונקציה קוראת את קובץ ה- structure ושומרת אותו (למעט את attribute class(מאפיין ה-class)) במילון מאפיינים. כאשר, ה-key הוא שם המאפיין והערך הוא רשימה של כלל האופציות אשר יכולות להיות במאפיין.
  לאחר מכן, הפונקציה רצה בלולאה על המילון ויוצרת שתי רשימות: רשימה אשר מחזיקה את כלל שמות המאפיינים הנומריים ורשימה אשר מחזיקה את כלל שמות המאפיינים הלא-נומריים(אנו נשתמש ברשימות האלו בהמשך בשביל להפריד בין התהליכים השונים, הללו אשר רלוונטיים לנומריים והללו אשר לא).
  בנוסף, הפונקציה שומרת את מאפיין ה-class ברשימה נפרדת.
- read_train: הפונקציה קוראת את קובץ ה- train ושומרת אותו במבנה נתונים מסוג רשימה של רשימות(list of lists) כאשר כל עמודה היא מאפיין.
- find_index: אשר שומרת את כל האינדקסים של המאפיינים בקובץ ה- train וזאת בכדי שנדע מול איזה סוג אנחנו עובדים, הפונקציה מרחיבה את שתי הרשימות של המאפיינים הנומריים ולא הנומריים.
- is_float: בשל מגבלה של פיתון 2 , בנינו פונקציה אשר מחזירה אמת אם משתנה הוא מספרי ושקר אם לא.
- str_to_num: הפונקציה מקבלת ערך מהרשומות הנומריות וממירה אותה במידת האפשר ל-float (היא נעזרת בis_float בשביל לקבוע אם ניתן לבצע את ההמרה). מקרים שבהם לא ניתן לבצע את ההמרה: ערך חסר או רעש .
- set_to_bins: הפונקצייה מקבלת ערך מהרשומות הנומריות ומחלקת אותה לבינים על פי כמות הבינים אשר המשתמש הכניס.
  הפונקציה משתמש בנוסחא $\left\lceil \frac{maxval - minval}{num\ of\ bins} \right\rceil$ בכדי למצוא את הטווח של כל בין.
  לשם הנוחות, במקום לשמור בזיכרון טווחים של כל בין ובין ולהעביר אליהם את הערכים, הפונקציה ממירה כל אחד ואחד מהערכים לערך המקסימלי של כל בין וזאת לפי הנוסחא הבאה: $\lceil value/range \rceil * range$.
- mean: פונקציה אשר מקבלת שורה המייצגת attribute נומרי על כל ערכיו ומחשבת את הערך הממוצע של השורה.

- fill_missing_numeric: הפונקציה מקבלת שורה המייצגת attribute נומרי ומשלימה בתוכו את החסרים(רעש הוא חוסר לכל דבר) על ידי הכנסת הערך הממוצע לתוך המקומות הנ"ל. לאחר מכן, הפונקציה מחלקת את הערכים לבינים.
הפונקציה נעזרת בפונקציית ה-mean בכדי לחשב את הערך הממוצע, בפונקציית str_to_num בכדי להמיר את המחרוזות לערכים נומריים ובפונקציית set_to_bins בכדי לבצע את החלוקה לבינים.

- mode: פונקציה אשר מקבלת שורה המייצגת attribute קטגוריאלי ומחשבת את הערך השכיח של השורה.
- fill_missing_non_numeric: הפונקציה מקבלת שורה המייצגת attribute קטגוריאלי ומשלימה בשורה את כלל הערכים החסרים(כולל החלפת רעש), הערכים מוחלפים בערך השכיח ביותר בשורה.
- non_numeric_preprocessing: הפונקציה נעזרת בפונקציה fill_missing_non_numeric בכדי להשלים את כל הערכים הקטגוריאליים אשר חסרים.
- calc_abs_n: פונקציה לחישוב כמות החזרות של כל class, אנו נעזר בפונקציה הזאת בחישוב הסטטיסטיקות ($|n|$).
- find_k_factor: בהתבסס על מבנה הנתונים, הפונקציה בונה מילון שכל key שלו הוא שם של מאפיין וכל value הוא כמות ה"אופציות" לכל מאפיין (K).
- findout_nc: הפונקציה מקבלת שורה המייצגת attribute ומחזירה מילון שכל key שלו הוא tuple המורכב מערך של המאפיין וה class שאליו הוא מוצמד בdataset , הערך הוא כמות החזרות של ה-tuple . למעשה, הפונקציה מחשבת את כמות החזרות של כל ערך בהינתן class מסוים (n_c).
- laplacian_estimator: הפונקציה מחשבת את הסטטיסטיקה של ערך מסוים ע"פ laplacian estimator , בעזרת הנוסחא: $\frac{n_c+1}{|n|+k}$
כאשר:
n_c-כמות החזרות של ערך מסוים בהינתן class מסוים.
$|n|$ -כמות החזרות של ה-class.
K - כמות ה"אופציות" של כל attribute.

- statistics_calculation: הפונקציה מקבלת שורה המייצגת attribute ומחשבת עבורה את כלל הסטטיסטיקות, הפונקציה נעזרת בפונקציות הנ"ל.
כל ריצה של הפונקציה מסתיימת בעדכון מילון הסטטיסטיקות (שבו ה- key זה שם ה-attribute וה-value הוא מילון שכל Key שלו הוא tuple המייצג ערך בהינתן class מסוים והערך הוא החישוב הסטטיסטי).
- transpose: הפונקציה מקבלת את ה- dataset והופכת אותו(שורות לעמודות ועמודות לשורות) וזאת בכדי שיהיה יותר נוח לעבוד עם ה-dataset.
הערה: הפונקציה עובדת לכל כמות שורות ועמודות(לא חייב להיות n*n).
- הפונקציות non_numeric ו—numericרצות על השורות הרלוונטיות אליהן(קטגוריאליות ונומריות בהתאמה) ומפעילות את כל הפונקציות לפי הסדר הרצוי (תחילה preprocessing ולבסוף חישוב סטטיסטיקות).
- update_observers:פונקציה אשר מעדכנת את ה-controller בהודעות(אשר דרך ה-controller עוברות להצגה בממשק הגרפי -view) ובנוסף, במקרה של סוף תהליך ישלחו כל הפרמטרים הרלוונטיים אשר ה-classify זקוק להם.

Classify

תהליך העבודה של ה-Classify

1. קריאת קובץ ה-test.
2. המרה של ערכים נומריים ממחרוזת לערך float (ללא בדיקה).
3. חלוקה של ערכים נומריים לbin-ים.
4. יצירת מילון אשר ה-key בו הוא שם ה- class והערך הוא 1(נבצע את ההכפלות לתוכו ובסוף נשלוף את הערך המקסימלי).
5. מעבר על כל שורה ושורה.
   5.1. בניית tuple-ים עם כל האופציות הקיימות עבור כל ערך וערך.
   5.2. בדיקה אם יש לנו סטטיסטיקות מתאימות:
   אם כן-נכפיל אותו לתוך המילון מסעיף 4.
   אם לא-נבצע חישוב סטטיסטי בדומה למה שבוצע ב-BuildClasifyer רק שבמקרה זה n_c יהיה שווה ל-0)
   5.3. בדיקת הערך המקסימלי במילון ורשימתו לתוך קובץ Output.txt.

פונקציות עיקריות ב-Classify :

- statistics_class: הפונקציה משנה את רשימת ה-class-ים הקיימים למילון שה-key בו הוא השם של ה-class והערך הוא 1.(בהמשך נכפיל לתוך המילון הזה את כל הסטטיסטיקות ובסוף נמצא את הערך המקסימלי).
- prepare:הפונקציה אחראית לניהול כל תהליך ההכנה של קובץ ה- test לסיווג, היא עושה זאת ע"י קריאה לפונקציות הרלוונטיות.
  1. תחילה, מתבצעת קריאה של הקובץ.
  2. לצורך הנוחות של חלוקה לבינים עבור ערכים נומריים, נבצע transpose.
  3. מחיקת השורה האחרונה(שורת ה-class-ים הצפויים).
  4. מעבר על כל השורות הנומריות.
     4.1. המרת הערכים ל-float(ללא בדיקה מקדימה).
     4.2. חלוקה לבינים(לפי הטווחים אשר התקבלו בתהליך הבנייה).
  5. Transpose.
- read_test: אשר קוראת את קובץ ה- test ושומרת אותו ברשימה של רשימות.
- Transpose: הפיכה של שורות לעמודות ולהפך.
- Classify:הפונקציה אחראית לניהול התהליך של סיווג השורה ורשימתה אל תוך קובץ.
  התהליך מתבצע על ידי השלבים הבאים:
  1. פתיחת קובץ output לכתיבה, הקובץ יפתח בתיקיה שבה נמצאים ה-dataset-ים.
  2. מעבר על כל שורה בלולאת for וסיווגה בהתאם.
     2.1. אתחול של מילון ה-class-ים אשר אחראי על שמירת הכפלת ההסתברויות(ההסתברות המותנית עבור כל class) בכל שורה ושורה.
     2.2. ריצה בלולאה בתוך כל שורה:
        2.2.1. קריאה לפונקציית mul אשר תכפיל את ההסתברויות.
        2.2.2. רישום ה-key עם הערך הגדול ביותר לקובץ.
- find_max: פונקציה אשר מחזירה אתה ה- key בעל הערך הגדול ביותר במילון ה-class-ים.
- Mul: פונקציה אשר מכפילה את הסטטיסטיקות, הפונקציה מקבלת את הדברים הבאים:

1. Tuple-ים המייצגים את כל האופציות האפשרויות של ההסתברות המותנית עבור הערך (למשל, אם הערך הוא:5 וה-class-ים שלנו הם:'no','yes' אז נקבל (5,'no'),(5,'yes').
2. האינדקס של המאפיין.

התהליך של הפונקציה:

1. תחילה הפונקציה רצה על רשימת ה-tuple-ים ובודקת אם קיים לנו חישוב ההסתברות עבור ה-tuple .
   1.1. במידה וכן, אז ניגשים ל- class המתאים במילון ה-class-ים ומכפילים לתוכו את ההסתברות.
   1.2. במידה ולא, מבצעים חישוב סטטיסטי לפי Laplacian estimator רק שהפעם n_c הוא 0.

- make_pairs: פונקציה אשר יוצרת את ה-tuple-ים המדוברים מעלה.
- update_observers:פונקציה אז מעבירה הודעות שוטפות על כל התקדמות ל-controller.
- str_to_num: פונקציה אשר מקבלת שורה המייצגת attribute נומרי וממירה את כל הערכים לערכים מספריים (בניגוד לפונקציה בBuildclassify הפונקציה הנ"ל לא בודקת אם ניתן להמיר את הערך- וזאת בשל ההנחות הרשומות בתחילת הקובץ).
- set_to_bins:פונקציה אשר מקבלת שורה המייצגת attribute נומרי ומחלק את הערכים לבינים וזאת על פי ה-טווחים אשר חושבו מקודם.

תבנית MVC

בכדי לשמור על המודולריות של התוכנית, הממשק הגרפי הופרד מהלוגיקה, הם אינם מכירים אחד את השני וכל התקשורת ביניהם מתבצעת על ידי ה-controller.

תבנית Observer

בכדי שהתוכנית תוכל לעדכן באופן יעיל את המשתמש על כל התקדמות של התהליך ,

ה-View הוגדר להיות Observer של ה-Controller וה-Controller להיות Observer של ה-model

ככה שבמידה ויש צורך בעדכון מסוים , ה-model מעדכן את ה- Controller וה-Controller את ה-View.

בכל עדכון (למעט, עדכון על סוף פעילות) ה-View ידפיס שתי Label-ים בתחתית המסך, העליון הוא ההודעה הקודמת והתחתון הוא ההודעה הנוכחית אשר התקבלה.

בעדכון על סיום פעילות:הלייבל התחתון יציג הודעה על סיום פעילות ואת הזמן אשר עבר מתחילת העבודה על ה-train. בנוסף, תופיע הודעה(Message Box) המורה על המשך הפעילות.