Laboration 1

Modelling and learning from data

Arvid Gramer September 2022

Models considered

- Gradient boosted classification:
  GDC is an ensemble method where multiple decision trees are trained one after each. The consecutive trees are trained with a stronger focus on the input that was misclassified by the previous tree.

- Random forest:
  It is also an ensemble method with a ’forest’ of decision trees. Every tree is trained on a random subset of the data and, if specified, with a random set of features from that data.

- Logistic regression:
  Is one of the oldest models, where a linear combination of the features are calculated to produce a number larger or smaller than 0. Using a sigmoid this is then mapped to either 1 or 0. The model is trained so that the coefficients in the linear combination mapped each set of input to the right prediction.

Models considered

Part 2

- K nearest neighbour:
  The model is maybe the most straight forward one and predicts that the output is the same as the K nearest training datapoints in the feature space to the input.

- Suport vector machine:
  The model is trained draw a line between the two classes using some data points that acts as boundary. Input data is the mapped to one of these region, where the prediction is made.

Data preprocessing

Encoding

- The data I considered qualitative was ’key’, ’mode’ and ’time_signature’. Since ’mode’ was either 1 or 0 it was already encoded, but ’key’ and ’time_signature’ was both one hot encoded. This was made since I did not interpret them to be ordinal, that is there was no obvious ranking between them and therefore not ideal to keep them as numbers since that implies that for example time_signature = 4 is more similar to time_signature = 5 and it the model would try to find a relation in this form. The data was instead encoded in a nominal fashion, where a set of features were created so that the feature was = 1 if it was of a certain type and all others were zero for that type.

Data preprocessing

Scaling

- For the more quantitative data, which was all other columns than the ones previously mentioned, I utilised standard scaling so that each column was mapped to having zero mean and unitary variance. This is made to make the data more easily digested by the numerical algorithms and minimise rounding errors.
- Before I scaled the data the column for duration was log-transformed. That is each number was swapped for the logarithm of that number. This was done since I figured it would be more interesting if a song was long or short, rather than how long or how short. Super long songs after the transform are just considered slightly larger than a long song, but a long song is still longer than a short song.

Feature engineering

- In order to win the competition I realised that I had to test some random, slightly crazy things, to part from the rest. I did this by creating to artificial features. It should be said that the given dataset was already fairly high level and engineered since most categories were already altered in some way. This left less room for new combinations.

- One that I tried was the product of duration and energy, and one was the product of acousticness and loudness. The previous one was an idea to catch how exhausting a song would be to listen to, if it was long and high energy it would be very tiresome. The other was an attempt to identify more or less rock-like songs. Neither of these improved performance noticeably.

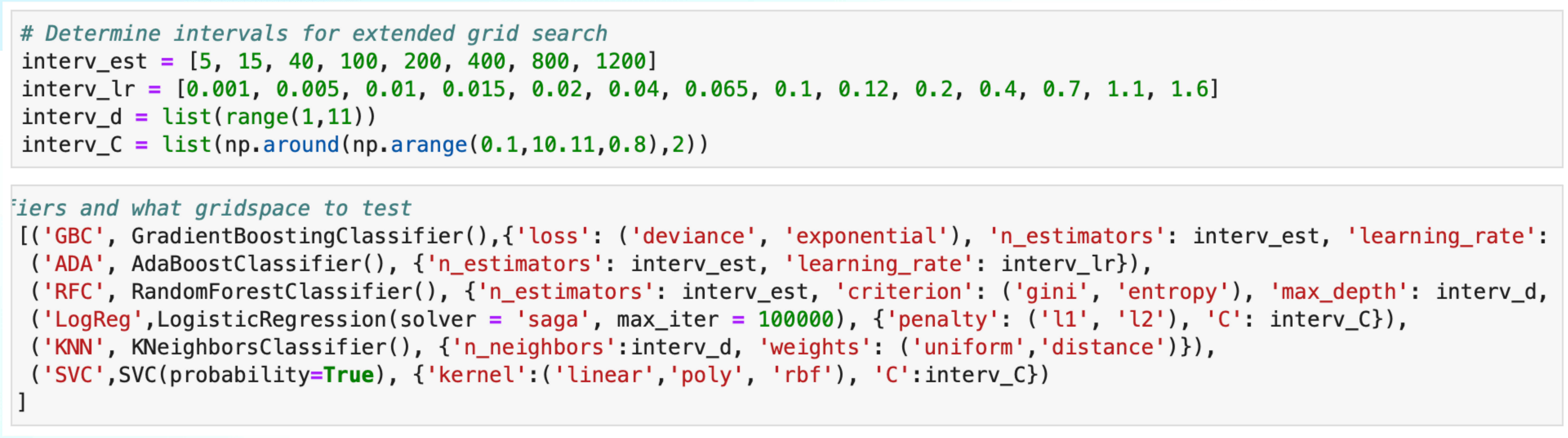
Feature engineering

Discarded tries

- I figured that the two features speechiness and instrumentalness should be each others opposites. A scatter plot of them showed that most songs were either *speechy* or *instrumental* so I tried to make a new feature of the ratio of these to distinguish them even further. This was not successful, it lowered performance and the idea was discarded.

Parameter tuning and validation

- I tuned the parameters using grid search over a quite wide grid of the most relevant parameters, and used the models that received the highest five fold cross validation score.

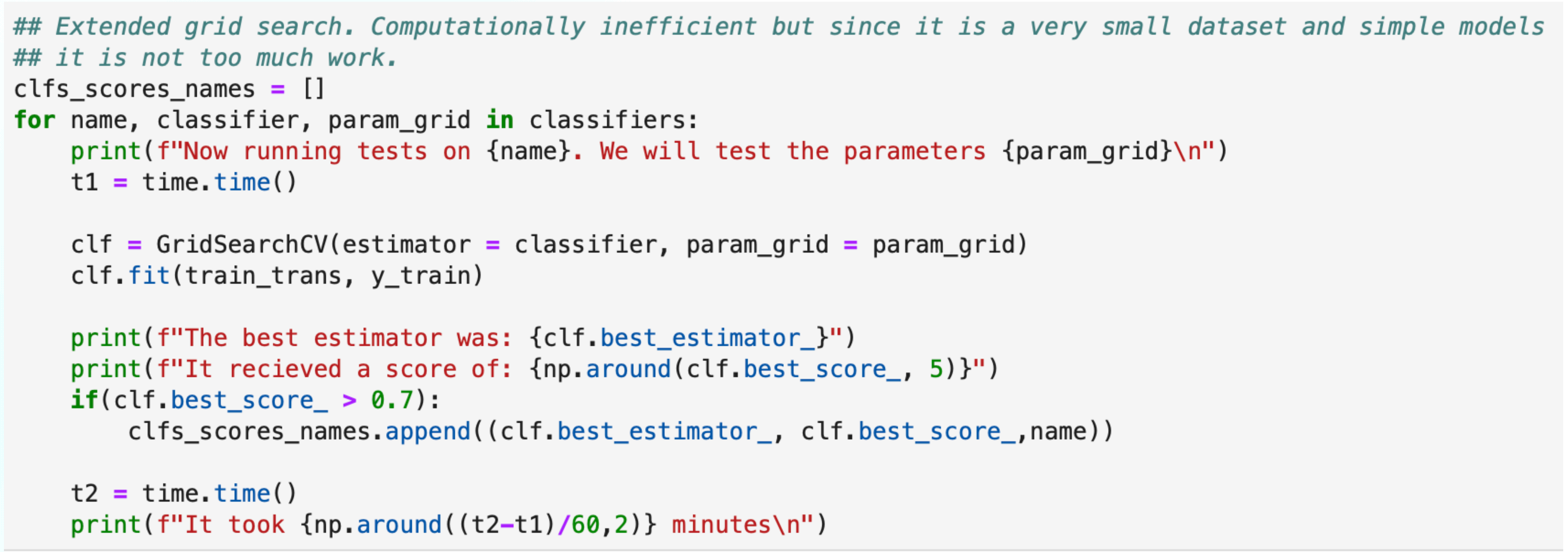


```
# Determine intervals for extended grid search
interv_est = [5, 15, 40, 100, 200, 400, 800, 1200]
interv_lr = [0.001, 0.005, 0.01, 0.015, 0.02, 0.04, 0.065, 0.1, 0.12, 0.2, 0.4, 0.7, 1.1, 1.6]
interv_d = list(range(1,11))
interv_C = list(np.around(np.arange(0.1,10.11,0.8),2))
```

```
iers and what gridspace to test
[('GBC', GradientBoostingClassifier(),{'loss': ('deviance', 'exponential'), 'n_estimators': interv_est, 'learning_rate':
 ('ADA', AdaBoostClassifier(), {'n_estimators': interv_est, 'learning_rate': interv_lr}),
 ('RFC', RandomForestClassifier(), {'n_estimators': interv_est, 'criterion': ('gini', 'entropy'), 'max_depth': interv_d,
 ('LogReg',LogisticRegression(solver = 'saga', max_iter = 100000), {'penalty': ('l1', 'l2'), 'C': interv_C}),
 ('KNN', KNeighborsClassifier(), {'n_neighbors':interv_d, 'weights': ('uniform','distance')}),
 ('SVC',SVC(probability=True), {'kernel':('linear','poly', 'rbf'), 'C':interv_C})
]
```

Final voting model

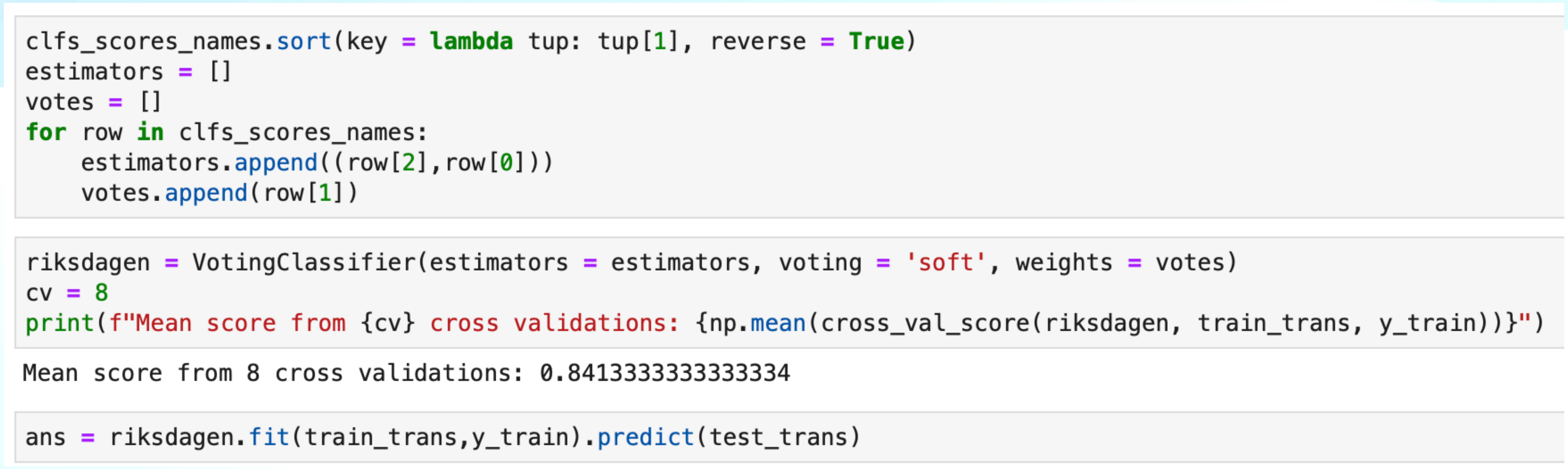
- I performed a quite extensive grid search over all my utilised models and different parameters, and added the best performing ones to a list together with their cross validation score. The final verdict on production data was then made through a soft vote, where each vote was weighted with that models validation score. This was made to reduce variance in the final model.



```python
## Extended grid search. Computationally inefficient but since it is a very small dataset and simple models
## it is not too much work.
clfs_scores_names = []
for name, classifier, param_grid in classifiers:
    print(f"Now running tests on {name}. We will test the parameters {param_grid}\n")
    t1 = time.time()

    clf = GridSearchCV(estimator = classifier, param_grid = param_grid)
    clf.fit(train_trans, y_train)

    print(f"The best estimator was: {clf.best_estimator_}")
    print(f"It recieved a score of: {np.around(clf.best_score_, 5)}")
    if(clf.best_score_ > 0.7):
        clfs_scores_names.append((clf.best_estimator_, clf.best_score_,name))

    t2 = time.time()
    print(f"It took {np.around((t2-t1)/60,2)} minutes\n")
```

Voting system

- The final model performed quite well on both validation (0.8413) and on test data (0.81) but not good enough to win sadly.



```
clfs_scores_names.sort(key = lambda tup: tup[1], reverse = True)
estimators = []
votes = []
for row in clfs_scores_names:
    estimators.append((row[2],row[0]))
    votes.append(row[1])
```

```
riksdagen = VotingClassifier(estimators = estimators, voting = 'soft', weights = votes)
cv = 8
print(f"Mean score from {cv} cross validations: {np.mean(cross_val_score(riksdagen, train_trans, y_train))}")
```

```
Mean score from 8 cross validations: 0.8413333333333334
```

```
ans = riksdagen.fit(train_trans,y_train).predict(test_trans)
```